OCTA

Integrated simulation system for soft materials

COARSE-GRAINED MOLECULAR DYNAMICS PROGRAM

COGNAC

USER'S MANUAL

Version 9.2

OCTA User's Group

Dec 25 2017

Authors of the Manual

chapter 1	Takeshi Aoyagi
chapter 2	Takeshi Aoyagi
chapter 3	Takeshi Aoyagi and Fumio Sawa
chapter 4	Takeshi Aoyagi and Fumio Sawa
chapter 5	Takeshi Aoyagi
chapter 6	Fumio Sawa
chapter 7	Takeshi Aoyagi
Appendix A	Takeshi Aoyagi
Appendix B	Takeshi Aoyagi
Appendix C	Takeshi Aoyagi and Fumio Sawa

Developers

Takeshi Aoyagi
Fumio Sawa
Tatsuya Shoji
Hiroo Fukunaga

Acknowledgment

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

Contents

List of Figures

List of Tables

Chapter 1

What is COGNAC?

COGNAC (COarse Grained molecular dynamics program by NAgoya Cooperation) is a program for general-purpose coarse-grained molecular dynamics simulations [**?**].

Molecular dynamics (MD) simulation has been a powerful tool for studying the microscopic structures and the properties of a system consisting of small molecules [**?**]. However, it is not so straightforward to do MD simulations for a system consisting of large and complex molecules such as polymers, surfactants and mesogens[**?**]. Characteristic time scales of such systems are usually much larger than those of small molecules, and often beyond the time scale which can be covered by the atomistic molecular dynamics. To extend the time and length scales, various non-atomistic models have been used such as the united atom model, bead-spring models for polymers, and ellipsoidal models for nematogenic molecules. These non-atomistic models are generally referred to as coarse-grained molecular models, and have been used extensively in the study of complex fluids [**?**].

COGNAC is developed as a part of OCTA. **COGNAC** aims at covering a large class of molecular models, ranging from full atomistic models to bead-spring models and dissipative particle dynamics (DPD). Such flexible molecular modeling is realized by a combination of choices of various potential functions between chemically connected pairs and chemically non-connected pairs. **COGNAC** is implemented with various dynamical equations of motion to study systems with constant temperature and/or pressure, under shear, under chemical reaction etc. Furthermore, **COGNAC** is written by the C++ code and designed for the easy extension of potential functions and equations of motions by users.

In addition to this versatile performance, **COGNAC** has special functions to deal with phase-separated systems. **COGNAC** can generate the equilibrium structures of phase-separated systems using the spatial distribution of each component which can be obtained by SUSHI and other programs. These functions can be used for bridging **COGNAC** with other meso-scale modeling programs such as **SUSHI** and **MUFFIN phaseseparation**.

In 2016, a textbook for OCTA[**?**] was published from Springer. There are many usuful informations for the theory and the application of OCTA system. The users of OCTA are recommended to refer the text book as well as this manual.

Chapter 2

Theoretical background

This chapter explains the theoretical background of the functions of **COGNAC**, and the algorithm for the (coarse-grained) molecular dynamics simulations used in **COGNAC**. The details of the functions are described in chapter 5. For the algorithm of molecular dynamics simulations, also refer to text books such as Ref.[**?**, **?**, **?**]

2.1 Outline of the functions

The main features of **COGNAC** are described below.

- Molecular models: Molecules of an arbitrary architecture, e.g. linear chains and branched chains, small molecules and single atoms, can be handled.
- Initial structures: Various initial structures, e.g. amorphous structure, crystal structure, semi-crystalline lamella structure and phase separated structures, can be generated.
- Molecular dynamics: NVE, NVT, NPH and NPT ensembles are available. In addition, non-equilibrium molecular dynamics with shear flow by the SLLOD algorithm or deformation of unit cell are available.
- Molecular mechanics: Energy minimization by the steepest descent and the conjugate gradient methods are implemented.
- DPD: Dissipative particle dynamics [**?**, **?**] is implemented.
- Potentials: Bonding potentials (bond, angle, torsion), non-bonding potentials, electrostatic potentials and external potentials such as a solid wall, a homogeneous electric field and a density biased potential are implemented.
- Boundary conditions: 2D and 3D periodic boundary conditions, Lees-Edwards boundary conditions and Staggered reflective boundary conditions are implemented.
- Chemical reactions: Creation and scission of chemical bonds can be handled.

2.2 Notes

In this section, some notes and definitions of terminology are described.

2.2.1 Unit system

COGNAC uses the reduced unit system for the input and output data. To convert it to the real unit, a set of unit parameters, which determine the reduced length, reduced mass and reduced time, must be given. The unit conversion can be easily done by setting the unit parameters of **COGNAC** UDF in GOURMET. See the section 5.2.9 and "GOURMET operation manual" for the details.

An example of the conversion of the reduced unit system (an united atom model)

In the case that
reduced length $1\sigma = 0.38$nm,
reduced energy (Energy density) $1\epsilon = 0.5$kJ/mol and
reduced mass $1m = 23.3 \times 10^{-27}$kg
(these values come from the mass of CH_2 unit and the parameter of the Lennard-Jones potential [**?**]),

other reduced quantities are obtained as follows;
reduced temperature $1T = \epsilon/\mathrm{R} = 60.1$K
reduced time $1t = \sqrt{\mathrm{Av}m\sigma^2/\epsilon} = 2.01$ps
reduced density $1\rho = m/\sigma^3 = 0.4246\mathrm{g/cm}^3$ and
reduced pressure $1P = \epsilon/(\mathrm{Av}\sigma^3) = 15.13$MPa,
where R is the gas constant and Av is the Avogadro's number.

2.2.2 Pressure and stress

In **COGNAC**, the sign of the pressure P and stress tensor σ are defined as follows. When the internal pressure P of a system is larger than the external pressure P_0, the system tends to expand. On the other hand, when the internal stress σ of a system is larger than the external stress σ_0, the system tends to compress. The relation between P and σ is given by $P = -\mathbf{Tr}\sigma/3$.

Moreover, when specifying the external pressure and the stress tensor at NPT ensemble etc. (refer to section 5.2.2), the pressure tensor of $P_0\mathbf{I} - \sigma_\mathbf{0}$ is applied to the system as an external pressure tensor.

2.2.3 Bending angle and torsion angle

In **COGNAC**, according to the definition often used in the polymer chemistry, **bending angle is defined as the outer angle, i.e. the bending angle is 0 degree for a straight molecule. The torsion angle is defined as 0 degree for the trans conformation**. The default unit for the angles is degree.

2.2.4 Terminology

Atom

Although one mass point does not correspond to one atom in the case of coarse-grained models, the word "atom" is used for all mass points in this manual.

Interaction site

COGNAC can handle not only spherical objects but also non-spherical ones such as ellipsoids. The objects are defined by one or more atoms as it is shown in Figure 2.1. We call them the interaction site. The pair interaction and the external interaction are calculated using the positions of atoms, which define the interaction site.

Electrostatic site

The electrostatic interaction can be treated independently from the non-bonding interactions. The interaction site for the electrostatic interaction is called as the electrostatic site. In the case of point charge, the electrostatic site is defined by one atom and in the case of dipole, it is defined by two atoms.

2.3 Molecular dynamics

The algorithm for the molecular dynamics used in **COGNAC** is explained.

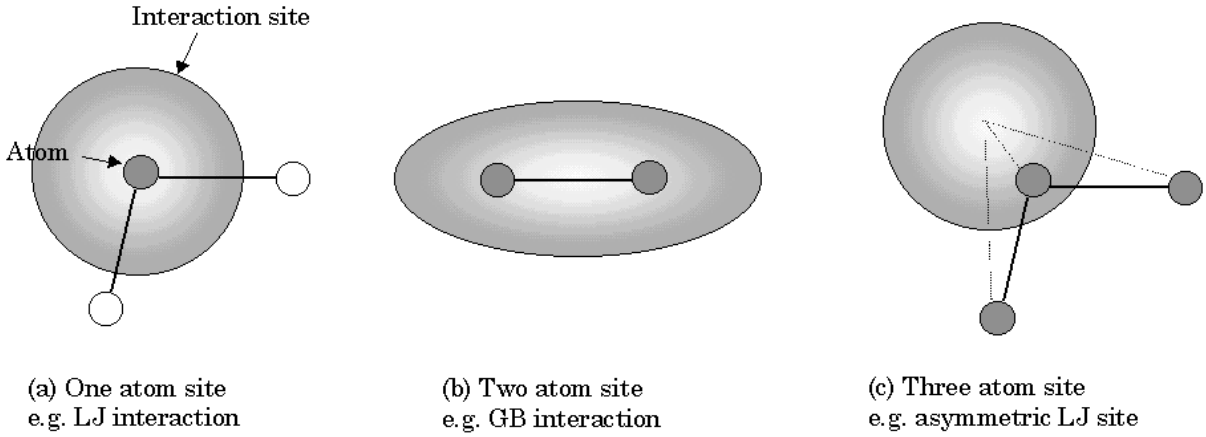

Figure 2.1: Definition of the interaction site

2.3.1 Calculation of temperature and pressure tensor

In molecular dynamics simulation, temperature and pressure tensor are calculated as follows,

- Temperature, T

$$T = \frac{1}{(3N - N_c)k_B} \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{m_i}, \tag{2.1}$$

where, N is a number of atoms, k_B is the Bolzman constant, $\mathbf{p}_i$ and m_i are the momentum and mass of atom i respectively. N_c is the number of constraint. Number 3 is added when 3D periodic boundary conditions are applied. Also, when the constraint condition described in the section 2.3.4 is applied, the number of constraint is added to N_c.

- Pressure tensor, $\mathbf{P}$

When the periodic boundary conditions or staggered reflective boundary conditions which is desribed in the section 2.8 is applied, the pressure tensor $\mathbf{P}$ is calculated by,

$$\mathbf{P} = \frac{1}{V}\left(\sum_{i=0}^{N} \frac{\mathbf{p}_i^2}{m_i} + \sum_{i=0}^{N-1}\sum_{j>i}^{N} \mathbf{r}_{ij}\mathbf{f}_{ij}\right), \tag{2.2}$$

where, V is the volume of the system, N is the number of atoms, $\mathbf{p}_i$ and m_i are the momentum and mass of atom i respectively, $\mathbf{r}_{ij}$ stands for $\mathbf{r}_i - \mathbf{r}_j$, $\mathbf{f}_{ij}$ is the force exerted on the atom i by the atom j.

2.3.2 Ensemble

- Micro canonical ensemble (NVE ensemble)

The number of atoms N, volume V, and the total energy E (the sum of potential energy and kinetic energy) of a system are kept constant. In the NVE ensemble, the temperature can be controlled by scaling the velocity:

$$\left(\frac{\mathbf{v}_{new}}{\mathbf{v}_{old}}\right)^2 = \frac{T_{target}}{T_{system}} \tag{2.3}$$

$\mathbf{v}_{new}$: velocity after scaling, $\mathbf{v}_{old}$: velocity before scaling
T_{target}: target temperature, T_{system}: current temperature

- Temperature controlling method (NVT ensemble)

  In addition to the velocity scaling mentioned above, **COGNAC** has the following algorithm for controlling the temperature.

  - Nose-Hoover [**?**]
    According to eqs.2.4–2.6, the virtual mass Q is given and the temperature is controlled.

$$\frac{\Delta \mathbf{q}_i}{\Delta t} = \frac{\mathbf{p}_i}{m_i} \tag{2.4}$$

$$\frac{\Delta \mathbf{p}_i}{\Delta t} = -\frac{\Delta U}{\Delta \mathbf{q}_i} - \zeta \mathbf{p}_i \tag{2.5}$$

$$\frac{\Delta \zeta}{\Delta t} = \frac{\sum_i \frac{\mathbf{p}_i^2}{m_i} - g k_B T}{Q} \tag{2.6}$$

    $\mathbf{q}_i$: real coordinate of an atom i, $\mathbf{p}_i$: momentum of an atom i, m_i: mass of an atom i, Δt: time step
    U: potential energy, g: degrees of freedom, T: target temperature, Q: fictitious mass.

  - Loose-coupling [**?**]
    The velocity of an atom is scaled by the scale factor λ given by,

$$\lambda = \left[1 + \frac{\Delta t}{\tau}\left(\frac{T_0}{T} - 1.0\right)\right]^{1/2} \tag{2.7}$$

    τ: characteristic relaxation time, T_0: target temperature, T: current temperature.

  - Kremer-Grest(The Langevin dynamics) [**?**]
    The temperature is controlled by the friction constant and the thermal noises. This is equivalent to the Langevin equation:

$$m_i \frac{d^2 \boldsymbol{r}_i}{dt^2} = \boldsymbol{F}_i - m_n \Gamma \frac{d\boldsymbol{r}_i}{dt} + \boldsymbol{W}_i(t) \tag{2.8}$$

    $\boldsymbol{r}_i$: coordinates of an atom i, m_n: mass of an atom i, $\boldsymbol{F}_i$: force acting on an atom i, Γ: friction constant, $\boldsymbol{W}_i(t)$: Gaussian white noise given by

$$\langle \boldsymbol{W}_i(t) \boldsymbol{W}_j(t') \rangle = 2 k_B T i_j \Gamma \delta_{ij} \boldsymbol{I} \delta(t - t'). \tag{2.9}$$

- Pressure controlling method (NPH ensemble)

  COGNAC has the following algorithm for controlling pressure.

  - Andersen [**?**]
    This algorithm uses the extended Lagrangian,

$$\mathcal{L} = \mathcal{K} + \mathcal{K}_V - \mathcal{V} - \mathcal{V}_V \tag{2.10}$$

    $\mathcal{K}$: kinetic energy of the system, $\mathcal{V}$: potential energy of the system, $\mathcal{K}_V$: kinetic energy of the additional variable, $\mathcal{V}_V$: potential energy of the additional variable.
    $\mathcal{K}_V$ and $\mathcal{V}_V$ are given by

$$\mathcal{K}_V = \frac{1}{2} Q \dot{V}^2 \tag{2.11}$$

$$\mathcal{V}_V = P_0 V \tag{2.12}$$

    Q: cell mass, V: volume, P_0: target pressure.

The equation of motion is given by

$$\ddot{\mathbf{s}} = \mathbf{f}/(mV^{1/3}) - (2/3)\dot{s}\dot{V}/V \quad (2.13)$$
$$\ddot{V} = (P - P_0)/Q \quad (2.14)$$

$\mathbf{s}$: scaled coordinate, i.e. $\mathbf{r} = V^{1/3}\mathbf{s}$ and $\mathbf{v} = V^{1/3}\dot{\mathbf{s}}$, $\mathbf{f}$: force, m: mass of atom, P: pressure.

– Parrinello-Rahman [**?**]
This algorithm is an extended version of the Andersen algorithm. Since in the Parrinello-Rahman algorithm, each components of the pressure tensor is controlled, this algorithm is often used to control the pressure in a crystalline structure where the deformation of the shape of the unit cell is needed.
The extended Lagrangian L is given by

$$\mathcal{L} = \mathcal{K} + \mathcal{K}_V - \mathcal{V} - \mathcal{V}_V \quad (2.15)$$

$\mathcal{K}$: kinetic energy of the system, $\mathcal{V}$: potential energy of the system, $\mathcal{K}_V$: kinetic energy of the additional variable, $\mathcal{V}_V$: potential energy of the additional variable.

$$\mathcal{K}_V = \frac{1}{2}Q\sum_\alpha\sum_\beta \dot{\mathbf{H}}_{\alpha\beta}^2 \quad (2.16)$$
$$\mathcal{V}_V = P_0 V \quad (2.17)$$

Q: cell mass, H: transformation matrix, i.e. $\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3)$, $\mathbf{h}_\alpha$: vector of each axis of the unit cell, V: volume, P: target pressure.

The equation of motion is given by

$$\ddot{\mathbf{s}} = \mathbf{H}^{-\mathbf{1}}\mathbf{f}/m - \mathbf{G}^{-1}\dot{\mathbf{G}}\dot{\mathbf{s}} \quad (2.18)$$
$$\ddot{\mathbf{H}} = (\mathbf{P} - \mathbf{1}P_0)V(\mathbf{H}^{-1})^{\mathrm{T}}/Q \quad (2.19)$$

$\mathbf{s}$: scaled coordinate $\mathbf{r} = \mathbf{Hs}$, $\mathbf{G} = \mathbf{H}^T\mathbf{H}$, $\mathbf{f}$: force, m: mass of atom, $\mathbf{P}$: pressure tensor.

When the angles of the unit cell are fixed, the off-diagonal components of $\ddot{\mathbf{s}}$ in eq.2.18 are forced to be 0. In this case, the Hamiltonian may not be conserved.

– Loose-coupling [**?**]
Each component of the pressure tensor is controlled by loose-coupling method. That is, an anisotropic deformation of a unit cell is possible like Parrinello-Rahman method.

- Temperature-pressure controlling method (NPT ensemble)

In **COGNAC**, some combinations of the temperature-controlling methods and the pressure-controlling method can be used.

– Andersen + Nose-Hoover
Temperature is controlled by the Nose-Hoover method, and pressure is controlled by the Andersen extended Hamiltonian method.

– Andersen + Kremer-Grest
Temperature is controlled by the Kremer-Grest method, and pressure is controlled by the Andersen extended Hamiltonian method.

– Parrinello-Rahman + Nose-Hoover
Temperature is controlled by the Nose-Hoover method, and pressure is controlled by the Parrinello-Rahman extended Hamiltonian method.

– Parrinello-Rahman + Kremer-Grest
Temperature is controlled by the Kremer-Grest method, and pressure is controlled by the Parrinello-Rahman extended Hamiltonian method.

– Loose-coupling [**?**]
Pressure and temperature are controlled by the Loose-coupling method. The isotropic and/or anisotropic deformations of a unit cell are possible. In the isotropic deformation, the pressure is controlled by only scaling the atomic coordinates and the unit cell size. The scaling parameter μ is given by

$$\mu = \left(1 + \frac{\Delta t}{\tau_p}\beta[P - P_0]\right)^{1/3} \tag{2.20}$$

Δt: time step, β: compressibility of the system, τ_p: characteristic relaxation time, P_0: target pressure, P: current pressure.

In the anisotropic deformation, the transformation matrix $\mathbf{H}$ is integrated according to eq.2.21 and the atomic coordinates are scaled by the transformation matrix.

$$\dot{\mathbf{H}} = \frac{\mathbf{P} - \mathbf{P_0}}{m} \tag{2.21}$$

m: cell mass, P_0: target pressure tensor, P: current pressure tensor.

When the angles of unit cell are fixed, the off-diagonal components of $\dot{\mathbf{H}}$ in eq.2.21 are forced to be zero.

- Non-equilibrium dynamics
COGNAC has the following two methods for performing non-equilibrium dynamics simulations.

Shear flow

COGNAC can simulate the system under shear flow. To do this, one uses NVE ensemble or the Langevin dynamics together with the Lees-Edwards boundary conditions[**?**]. The temperature is controlled by velocity scaling or by the Langevin dynamics. The temperature can also be controlled by the SLLOD with Nose-Hoover method[**?**]. The equation of motion of the SLLOD under the shear flow is given by [**?**]

$$\dot{r}_{ix} = p_{ix}/m_i + \dot{\gamma} r_{iy} \tag{2.22}$$
$$\dot{r}_{iy} = p_{iy}/m_i \tag{2.23}$$
$$\dot{r}_{iz} = p_{iz}/m_i \tag{2.24}$$
$$\dot{p}_{ix} = f_{ix} - \dot{\gamma} p_{iy} \tag{2.25}$$
$$\dot{p}_{iy} = f_{iy} \tag{2.26}$$
$$\dot{p}_{iz} = f_{iz} \tag{2.27}$$

$\dot{\gamma}$: shear rate, $r_{i\alpha}$: coordinate of an atom i, $p_{i\alpha}$: moment of an atom i, $f_{i\alpha}$: force on an atom i, m_i: mass of an atom i.

In addition to the temperature control, stress control is also possible in **COGNAC**. The component of the stress in the normal direction is controlled by the Parrinello-Rahman method, i.e. in the case of shear flow with the shear rate $\dot{\gamma}_{yx}$, only a length of c-axis of the unit cell is changed in order to control the component of the stress of the z-direction.

Note: The algorithm for temperature and stress control are changed to Nose-Hoover and Parrinello-Rahman algorithm respectively in Version 8.0.

Cell deformation

COGNAC can simulate the system under deformation. Both stress-controlled deformation and strain-controlled deformation are available.

- Stress-controlled deformation
  If the external stress is given, the time evolution in the deformation of the unit cell can be calculated by the Parrinello-Rahman method or the loose-coupling method.

- Strain-controlled deformation
  If the external strain is given, a unit cell size is deformed at a certain interval of the time step. In this case, it is necessary to select the interval of deformation properly. If this interval of the deformation is taken too long, the amount of deformations will become too large, and the simulation will become unstable. On the contrary, the simulation needs longer CPU time, if the interval of the deformation is taken too short.
  COGNAC can set a displacement tensor or an extensional rate in the deformation method.

2.3.3 Integration algorithm

In **COGNAC**, the velocity Verlet algorithm[**?**] is used for the time integration of MD.
This algorithm has advantages in the numerical stability and the accuracy of the conservation of Hamiltonian. In the velocity Verlet algorithm, the coordinate $\mathbf{r}$ and the velocity $\mathbf{v}$ at a time $t + \Delta t$ are determined by

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t) + \frac{1}{2}\Delta t^2 \mathbf{a}_i(t) \tag{2.28}$$

$$\mathbf{v}_i(t+\frac{1}{2}\Delta t) = \mathbf{v}_i(t) + \frac{1}{2}\Delta t \mathbf{a}_i(t) \tag{2.29}$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t+\frac{1}{2}\Delta t) + \frac{1}{2}\Delta t \mathbf{a}_i(t+\Delta t) \tag{2.30}$$

$\mathbf{a}_i$:acceleration, Δt:time step

2.3.4 Constraint

- RATTLE
  To fix bond lengths and bending angles, the RATTLE algorithm [**?**] is implemented in **COGNAC**.

- Constraint atom
  COGNAC has a function to constrain the velocity of specified atoms to constant value. If the velocity is set to 0, the atom is fixed during the simulation.

2.3.5 Energy flow

COGNAC can calculate the energy that flows between the system and the surrounding during the MD simulation. This function is only available in the case of the NVE ensemble with the velocity scaling method. When a temperature was scaled according to eq.2.3, the energy flow is given by

$$Energy\_Flow = \frac{1}{2} N k_B (T_{system} - T_{target}) \tag{2.31}$$

N: degrees of freedom of the system

The energy flow become negative if the energy is supplied to the system from the surrounding.

2.4 DPD

2.4.1 Equation of motions

The motion of Atom i in DPD is governed by Newton's equation of motion

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i, \quad m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{f}_i \tag{2.32}$$

The force acting on the atom i, $\mathbf{f}_i$ contains three parts,

$$\mathbf{f}_i = \sum_{j \neq i} \left(\mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R \right) \tag{2.33}$$

where $\mathbf{F}_{ij}^C$ is the conservative force, and $\mathbf{F}_{ij}^D$ and $\mathbf{F}_{ij}^R$ are a dissipative force or drag force and a random force. They are given by

$$\mathbf{F}_{ij}^D = -\gamma w^D(r_{ij})(\hat{\mathbf{r}}_{ij} \cdot \mathbf{v}_{ij})\hat{\mathbf{r}}_{ij}, \quad \mathbf{F}_{ij}^R = \sigma w^R \theta_{ij} \hat{\mathbf{r}}_{ij} \tag{2.34}$$

where $\theta_{ij}(t)$ is a randomly fluctuating variable with Gaussian statistics: $\langle \theta_{ij}(t) \rangle = 0$ and $\langle \theta_{ij}(t)\theta_{kl}(t') \rangle = (\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})\delta(t - t')$.

According to the study by Groot and Warren [**?**], the relation of other variables is as follows,

$$w^D(r) = [w^R(r)]^2 = \begin{cases} (1 - r/r_c)^2 & (r < r_c) \\ 0 & (r \geq r_c) \end{cases} \tag{2.35}$$

r_c: cutoff distance ($r_c = 1$ in the reference)

$$\sigma^2 = 2\gamma k_B T \tag{2.36}$$

2.4.2 Integration algorithm

The modified version of the velocity Verlet (2.37-2.40) is used for the time integration.

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t) + \frac{1}{2}\Delta t^2 \mathbf{f}(t)/m_i \tag{2.37}$$

$$\tilde{\mathbf{v}}_i(t + \Delta t) = \mathbf{v}_i(t) + \lambda \Delta t \mathbf{f}_i(t)/m_i \tag{2.38}$$

$$\mathbf{f}_i(t + \Delta t) = \mathbf{f}_i(\mathbf{r}_i(t + \Delta t), \tilde{\mathbf{v}}_i(t + \Delta t)) \tag{2.39}$$

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \frac{1}{2}\Delta t(\mathbf{f}_i(t) + \mathbf{f}_i(t + \Delta t))/m_i \tag{2.40}$$

λ : variable factor

2.5 Molecular mechanics

COGNAC has the function for energy minimization. For the details of algorithm of the optimization , refer to textbooks.

2.5.1 Optimization algorithm

- Steepest descent method
- Conjugate gradient method
- Cascade

  A combination of steepest descent and conjugate gradient. The steepest descent method is used in the initial stage, and the method is swithed to the conjugate gradient method after the gradient of potential decreased.

2.6 Potential functions

COGNAC calculates the potential energy and the forces acting on each atom in the system by the following interaction potentials.

$$U_{pot} = U_{bond} + U_{angle} + U_{torsion} + U_{non-bonding} + U_{elecrostatic} + U_{external} \tag{2.41}$$

U_{bond} : bond stretching potentials (two body potentials)
U_{angle} : angle bending potentials (three body potentials)
$U_{torsion}$: torsion angle rotational potentials (four body potentials)
$U_{non-bonding}$: non-bonding pair potentials
$U_{electrostatic}$: electrostatic potentials
$U_{external}$: external field potentials

These potential functions are explained below.

2.6.1 Bond stretching potentials

The bond stretching potential is a function of the distance between the chemically bonded atoms r. The following functions are available.

- Harmonic

$$U_{bond}(r) = \frac{1}{2}k(r - r_0)^2 \tag{2.42}$$

  k: spring constant, r_0: equilibrium bond length

- FENE+LJ [**?**]

$$U_{bond}(r) = U_{FENE}(r) + U_{LJ}(r) \tag{2.43}$$

$$U_{FENE}(r) = \begin{cases} -\frac{1}{2}kR_0^2 \ln\left[1 - \left(\frac{r}{R_0}\right)^2\right], & r < R_0 \\ \infty, & r \geq R_0 \end{cases} \tag{2.44}$$

  k: spring constant, R_0: finite extended length

$$U_{LJ}(r) = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6 + \frac{1}{4}\right], & r < 2^{1/6}\sigma \\ 0, & r \geq 2^{1/6}\sigma \end{cases} \tag{2.45}$$

  σ: diameter of the LJ sphere, ϵ: strength of the interaction

  Note: The output of energy of FENE+LJ potential has changed to the exact value of the potential function in the version 8, while it was arbitrary shifted before the version 7.

- Gaussian

$$U_{bond}(r) = \frac{3k_BTr^2}{2r_0^2} \tag{2.46}$$

k_B: Boltzmann constant, T: temperature, r_0: equilibrium bond length

- Morse
  - In the case that minimum energy is set zero,

$$U_{bond}(r) = A[exp\{-B(r - r_0)\} - 1]^2 \tag{2.47}$$

  - In the case that energy at infinit distance is set zero,

$$U_{bond}(r) = A[exp\{-B(r - r_0)\} - 1]^2 - A \tag{2.48}$$

A, B: constants, r_0: equilibrium bond length

- Bond polynomial

$$U_{bond}(r) = \sum_{n=0}^{N-1} A_n r^n \tag{2.49}$$

$N - 1$: order of polynomial, A_n: constants

- Table

  When the energy U_{bond} is tabulated as a function of r or r and target temperature T, the energies and forces are calculated by the interpolation scheme during simulations. If Fast_Table is selected, only r is a variable of the energy table.

- DPD

  Bond potential which is used in the DPD simulation of Groot et al [**?**, **?**]. This is essentially the same as the harminic potential.

$$U_{bond}(r) = \frac{1}{2}Cr^2 \tag{2.50}$$

C: constant

- User bond

  Users can use their own potential functions which are defined in the source code level.

2.6.2 Angle bending potentials

The angle bending potential is a function of the angle θ which is defined by the three chemically connected atoms. The following functions are available.

- Theta harmonic

$$U_{angle}(\theta) = \frac{1}{2}k(\theta - \theta_0)^2 \tag{2.51}$$

k: spring constant, θ_0: equilibrium angle

- Theta harmonic2

$$U_{angle}(\theta) = k\{1 - \cos(\theta - \theta_0)\} \tag{2.52}$$

k: spring constant, θ_0: equilibrium angle
Since the calculation of the forces diverge when θ becomes 0 or $180deg.$ in eq.2.51, it is recommended to use this equation in that case.

- Cosine harmonic

$$U_{angle}(\theta) = \frac{1}{2}k(\cos\theta - \cos\theta_0)^2 \tag{2.53}$$

k: spring constant, θ_0: equilibrium angle

- Theta polynomial

$$U_{angle}(\theta) = \sum_{n=0}^{N-1} A_n\theta^n \tag{2.54}$$

$N-1$: order of polynomial, A_n: constants

- Table

When the energy U_{angle} is tabulated as a function of θ or θ and target temperature T, the energies and forces are calculated by the interpolation scheme during simulations. If Fast_Table is selected, only *theta* is a variable of the energy table.

- User angle

Users can use their own potential functions which are defined in the source code level.

2.6.3 Torsion angle potentials

The torsion angle potential is a function of the torsion angle ϕ which is defined by the four chemically connected atoms. The following functions are available.

- Cosine polynomial

$$U_{torsion}(\phi) = k\sum_{n=0}^{N-1} A_n\cos^n\phi \tag{2.55}$$

k: constant, $N-1$: order of polynomial, A_n: constants

- Amber [**?**]

$$U_{torsion}(\phi) = \frac{PK}{IDIVF}\{1 + \cos(PN\phi - PHASE)\} \tag{2.56}$$

PK: one-half of the barrier magnitude, $IDIVF$: total number of torsions about a single bond, PN: periodicity, $PHASE$: phase shift

- Dreiding [**?**]

$$U_{torsion}(\phi) = \frac{V}{2}[1 - \cos\{n(\phi - phi0)\}] \tag{2.57}$$

V: barrier magnitude, n: periodicity, $phi0$: equilibrium angle

- Table

When the energy $U_{torsion}$ is tabulated as a function of ϕ or ϕ and target temperature T, the energies and forces are calculated by the interpolation scheme during simulations. If Fast_Table is selected, only phi is a variable of the energy table.

- User torsion

Users can use their own potential functions which are defined in the source code level.

2.6.4 Non-bonding pair potentials

The non-bonding pair potentials are calculated based on the interaction site defined in **COGNAC** as mentioned above. The following functions are available.

- Lennard-Jones

$$U_{nonbond}(\mathrm{r}_{ij}) = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] + U_{cutoff} & r_{ij} < r_c \\ 0, & r_{ij} \geq r_c, \end{cases} \tag{2.58}$$

σ: diameter of the LJ sphere, ϵ: strength of the interaction r_c: cutoff distance

  - When **COGNAC** uses the option for the tail correction, the term U_{cutoff} is not added.
  - When the tail correction is not performed, the term U_{cutoff} is chosen so that the $U_{nonbond}(r_{ij})$ becomes 0 at the cutoff distance.

When an interaction site for the Lennard-Jones potential is defined by two atoms, the interaction center is set at the mean position of the two atoms.

- Lennard-Jones with excluded volume [**?**]

$$U_{nonbond}(\mathrm{r}_{ij}) = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r_{ij}-R_{EV}}\right)^{12} - \left(\frac{\sigma}{r_{ij}-R_{EV}}\right)^{6}\right] + U_{cutoff}, & r_{ij} < r_c, \\ 0, & r_{ij} \geq r_c, \\ \infty, & r_{ij} \leq R_{EV}. \end{cases} \tag{2.59}$$

σ: diameter of the LJ sphere, ϵ: strength of the interaction, R_{EV}: trancated length, r_c: cutoff distance

- General Lennard-Jones

$$U_{nonbond}(\mathrm{r}_{ij}) = \begin{cases} \epsilon_{ij}\left[A\left(\frac{\sigma_0}{r_{ij}}\right)^{m} - B\left(\frac{\sigma_0}{r_{ij}}\right)^{n}\right] + U_{cutoff}, & r < r_c, \\ 0, & r \geq r_c, \end{cases} \tag{2.60}$$

σ: diameter of the LJ sphere, ϵ: strength of the interaction, m: power of repulsive term n: power of dispersive term r_c: cutoff distance

- Gay-Berne [?, ?]

  The Gay-Berne potential represents non-bonding potential between two ellipsoidal objects.

$$U_{nonbond}(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j, \mathrm{r}_{ij}) = 4\epsilon(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) \left[\left(\frac{\sigma_0}{\mathrm{r}_{ij} - \sigma(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) + \sigma_0} \right)^{12} - \left(\frac{\sigma_0}{\mathrm{r}_{ij} - \sigma(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) + \sigma_0)} \right)^{6} \right] \tag{2.61}$$

$$\sigma(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) = \sigma_0 \left[1 - \frac{\chi}{2} \left\{ \frac{(\alpha(\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_i) + \alpha^{-1}(\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_j))^2}{1 + \chi(\hat{\mathrm{u}}_i \cdot \hat{\mathrm{u}}_j)} + \frac{(\alpha(\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_i) - \alpha^{-1}(\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_j))^2}{1 - \chi(\hat{\mathrm{u}}_i \cdot \hat{\mathrm{u}}_j)} \right\} \right]^{-1/2} \tag{2.62}$$

$$\sigma_0 = \sqrt{d_i^2 + d_j^2} \tag{2.63}$$

$$\chi = \left[\frac{(l_i^2 - d_i^2)(l_j^2 - d_j^2)}{(l_j^2 + d_i^2)(l_i^2 + d_j^2)} \right]^{1/2} \tag{2.64}$$

$$\alpha^2 = \left[\frac{(l_i^2 - d_i^2)(l_j^2 + d_i^2)}{(l_j^2 - d_j^2)(l_i^2 + d_j^2)} \right]^{1/2} \tag{2.65}$$

$$\epsilon(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) = \epsilon_0 \epsilon_1(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j)^\nu \epsilon_2(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij})^\mu \tag{2.66}$$

$$\epsilon_1(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j) = \left[1 - \chi^2 (\hat{\mathrm{u}}_i \cdot \hat{\mathrm{u}}_j)^2 \right]^{-1/2} \tag{2.67}$$

$$\epsilon_2(\hat{\mathrm{u}}_i, \hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) = 1 - \frac{\chi'}{2} \left\{ \frac{(\alpha'(\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_i) + \alpha'^{-1}(\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_j))^2}{1 + \chi'(\hat{\mathrm{u}}_i \cdot \hat{\mathrm{u}}_j)} + \frac{(\alpha'(\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_i) - \alpha'^{-1}(\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_j))^2}{1 - \chi'(\hat{\mathrm{u}}_i \cdot \hat{\mathrm{u}}_j)} \right\} \tag{2.68}$$

$$\chi' = \frac{k'^{1/\mu} - 1}{k'^{1/\mu} + 1} \tag{2.69}$$

  – In practice, the term of U_{cutoff} was added automatically, and $U_{nonbond}$ is shifted automatically so that it may become 0 at cutoff distance. There is no function for tail correction.

  – Notice that σ_0 is taken from the input value and it is not calculated from d_i and d_j.

- Gay-Berne - Lennard-Jones pair [?]

  The Gay-Berne – Lennard-Jones potential represents the non-bonding potential between one ellipsoidal object and one spherical object.

$$U_{nonbond}(\hat{\mathrm{u}}_j, \mathrm{r}_{ij}) = 4\epsilon(\hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) \left[\left(\frac{\sigma_0}{\mathrm{r}_{ij} - \sigma(\hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) + \sigma_0} \right)^{12} - \left(\frac{\sigma_0}{\mathrm{r}_{ij} - \sigma(\hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) + \sigma_0} \right)^{6} \right] \tag{2.70}$$

$$\sigma(\hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) = \sigma_0 \left[1 - \chi \alpha^{-2} (\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_j)^2 \right]^{-1/2} \tag{2.71}$$

$$\epsilon(\hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) = \epsilon_0 \epsilon_2(\hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij})^\mu \tag{2.72}$$

$$\epsilon_2(\hat{\mathrm{u}}_j, \hat{\mathrm{r}}_{ij}) = 1 - \chi' \alpha'^{-2} (\hat{\mathrm{r}}_{ij} \cdot \hat{\mathrm{u}}_j)^2 \tag{2.73}$$

$$\sigma_0 = \sqrt{d_i^2 + d_j^2} \tag{2.74}$$

$$\frac{\chi}{\alpha^2} = \frac{l_j^2 - d_j^2}{l_j^2 + d_j^2}, \qquad \chi' \alpha'^{-2} = 1 - k'^{1/\mu} \tag{2.75}$$

  – Using Gay-Berne–Lennard-Jones scheme, potential calculations are performed by $\alpha = \alpha' = 1.0$ actually.

  – In practice, the term of U_{cutoff} was added automatically, and $U_{nonbond}$ is shifted automatically so that it may become 0 at cutoff distance. There is no function of tail correction.

– Notice that σ_0 is taken from the input value and it is not calculated from d_i and d_j.

- Table

  When the energy $U_{non-bonding}$ is tabulated as a function of r_{ij} or r_{ij} and target temperature T, the energies and forces are calculated by the interpolation scheme during simulations. If Fast_Table is selected, only r_{ij} is a variable of the energy table.

  – If the flag of tail correction is turned off, the term of U_{cutoff} will be added, and $U_{nonbond}$ is shifted automatically so that it may become 0 at the cutoff distance. However, since there is no function of tail correction, the correction is not carried out even in the case of the flag on.

- DPD

  Nonbonding potential which is used in the DPD simulation of Groot et al [**?**, **?**].

$$U_{nonbond}(\mathrm{r}_{ij}) = \begin{cases} \frac{1}{2}a_{ij}r_c(1 - r_{ij}/r_c)^2 & r_{ij} < r_c \\ 0, & r_{ij} \geq r_c, \end{cases} \tag{2.76}$$

  r_c: cutoff distance ($r_c = 1$ in the reference)

- Morse

$$U_{nonbond}(r) = A[exp\{-B(r - r_0)\} - 1]^2 - A \tag{2.77}$$

  A, B: constants, r_0: equilibrium length

- Buckingham

$$U_{nonbond}(r) = Aexp(-Br) - \frac{C}{r^6} \tag{2.78}$$

  A, B, C: constants

- User pair potential

  Users can use their own potential functions which are defined in the source code level.

Tail correction

In the calculation of the non-bonding pair interaction, the cutoff distance r_{cutoff} is generally set and the interaction of the distance larger than r_{cutoff} is not calculated. Tail correction is performed in order to correct the energy and the force disregarded by this cutoff. Although the dynamics is not affected by the correction of energy, the correction of force is important when the simulation is performed under constant-pressure conditions.
When the tail correction is adopted, the energy U_{corr} and the pressure P_{corr} derived from the pairs beyond the cutoff distance are corrected as follows:

$$U_{corr} = 2\pi N\rho \int_{r_{cutoff}}^{\infty} r^2 U(r) dr \tag{2.79}$$

$$P_{corr} = (2/3)\pi N\rho \int_{r_{cutoff}}^{\infty} r^2 \frac{r dU(r)}{dr} dr \tag{2.80}$$

N: number of atom, ρ: density

The tail correction is available only in the Lennard-Jones and general Lennard-Jones potential.

2.6.5 External potential

In **COGNAC**, the external potential, such as a homogeneous field and a solid wall can be applied. The following functions are available.

- Lennard-Jones type atomic wall

  The solid wall is represented by a set of atoms placed on the square lattice on the surface of a wall, and the interaction between an atom and the wall is obtained as the summation of the interactions between the atom and the wall atoms.

- Lennard-Jones type flat wall [**?**]

  In this type of interaction, the wall atoms are assumed to be distributed homogeneously with the density ρ, and the summation over wall atoms on the wall is replaced by the integration. This gives the following potential for the atom which is located at the distance r from the wall.

$$U_{ext}(\mathrm{r}) = 4\pi\rho_{wall}\epsilon_{wall}\left[\frac{1}{5}\left(\frac{\sigma_{wall}}{\mathrm{r}}\right)^{10} - \frac{1}{2}\left(\frac{\sigma_{wall}}{\mathrm{r}}\right)^{4}\right] + U_{cutoff} \tag{2.81}$$

  σ_{wall}: Diameter of the LJ sphere, ϵ_{wall}: Strength of the interaction, ρ_{wall}: density of wall

  This function is available only for a rectangular cell.

  Figure 2.2 shows the comparison between the LJ type atomic wall and the LJ type flat wall.

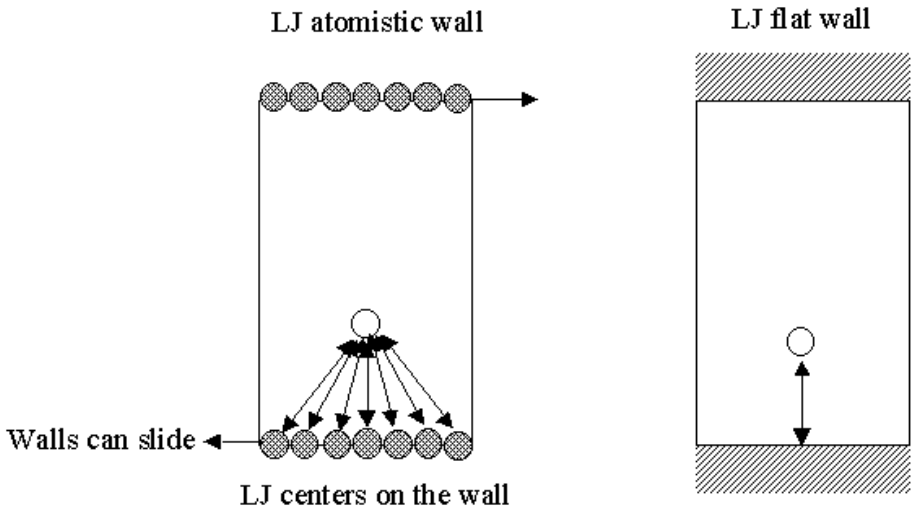

Figure 2.2: Comparison between LJ atomic wall and LJ type flat wall

When the wall potentials are applied, the wall pressure, $\mathbf{P}^{wall}$ is calculated by

$$\mathbf{P}^{wall} = \frac{1}{A}\sum_{i=1}^{N}\mathbf{f}_i^{wall}, \tag{2.82}$$

where, N is total number of atoms, A is the area of wall, and $\mathbf{f}_i^{wall}$ is the force acting on atom i by the wall potential. In the case of flat wall, $\mathbf{f}_i^{wall}$ has a none-zero value of components only perpendicular to the wall. On the other hand, atomistic wall has none-zero value of components parallel to the wall too.

- Density biased potential[**?**, **?**]

  COGNAC can read the spatial distribution of each atom in a system obtained by the mean field calculation program **SUSHI** and **Muffin_phaseseparation**, and evaluates the potential $U_{ext}(\mathbf{r})$ acting on the atom placed in the same mean field.

  SUSHI and **Muffin_phaseseparation** give the spatial distributions of segments in the phase-separated state. Let $\phi_n(\boldsymbol{r})$ be the local volume fraction of segment type n, $\phi_n(\boldsymbol{r})$ being the probability that a point $\boldsymbol{r}$ in a space is occupied by the segment type n. In numerical calculations, $\phi_n(\boldsymbol{r})$ is given as $\phi_n(i,j,k)$, where (i,j,k) is the index of the lattice point and $\phi_n(\boldsymbol{r})$ is calculated by the interpolation from the value of $\phi_n(i,j,k)$ of the nearby lattice points.

  If a segment of the SCF calculation corresponds to an atom of the MD simulation, the density biased potential acting on the segment (atom) type m at position $\boldsymbol{r}$ is described as follows,

$$U_{ext}(\boldsymbol{r}_m) = k_B T \sum_n \chi_{mn} \phi_n(\boldsymbol{r}_m), \tag{2.83}$$

  where χ_{mn} is an interaction parameter between segment type m and n used in the mean field calculation.

  From **COGNAC** version 4.2, the **Grid_Density** output of **COGNAC** can be read for the density field potential. For example, the density field data obtained by DPD calculation can be used for the exteral field acting on other model such as a bead-spring model.

- Lennard Jones type density oriented potential

  The external field is based on the density distribution on regular mesh, which is obtained by **SUSHI** and **Muffin_phaseseparation**. The potential formula is shown in eq.2.84, and similar to the Lennard-Jones type potential. The volume fraction on the position of atom $\boldsymbol{r}$, $\phi\boldsymbol{r}$ is related to the distance in the Lennard-Jones type potential.

$$U_{ext}(\boldsymbol{r}) = \begin{cases} 4\epsilon \left[\left(\frac{\sigma_0}{(1-\phi(\boldsymbol{r}))L} \right)^{12} - \left(\frac{\sigma_0}{(1-\phi(\boldsymbol{r}))L} \right)^{6} \right] & 0 < \phi(r) < 1.0 \\ 0, & \phi(r) \leq 0, \end{cases} \tag{2.84}$$

  σ : diameter of the LJ sphere, ϵ : strength of the interaction L : grid spacing

- Reciprocal power type density oriented potential

  The external field is based on the density distribution on regular mesh, which is obtained by **SUSHI** and **Muffin_phaseseparation**. The potential formula is shown in eq.2.85. Only repulsive force is applied to the atom.

$$U_{ext}(\boldsymbol{r}) = \begin{cases} \frac{k}{L^n(1-\phi(\boldsymbol{r}))^n} & 0 < \phi(r) < 1.0 \\ 0, & \phi(r) \leq 0, \end{cases} \tag{2.85}$$

  k : constant n : order of power L : grid spacing

- Total density constrain

  To constrain the total density of atoms, the external potential field corresponding to the current distribution of the atoms can be applied by

$$U_{ext}(\mathbf{r}) = k\phi(\mathbf{r}) \tag{2.86}$$

  k: coefficient, $\phi(\mathbf{r})$: total volume fraction at position $\mathbf{r}$.

  The volume fraction on the lattice point is calculated from the distribution of atoms by the method explained in the section 2.10, and $\phi(\mathbf{r})$ is calculated from the volume fraction $\phi(i,j,k)$ on the lattice

point.

- External angle potential

  To constrain the angle of three atoms, the bending potential of cosine harmonic type is applied to the angles which exist only in a specified region. It can be used to generate the partially ordered structures such as semi-crystalline lamella.

- External torsion potential

  To constrain the torsion angle of four atoms, the torsion potential of cosine polynomial type is applied to the torsion which exist only in a specified region. It can be used to generate the partially ordered structures such as semi-crystalline lamella.

  Figure2.3 illustrates the external angle/torsion potentials.

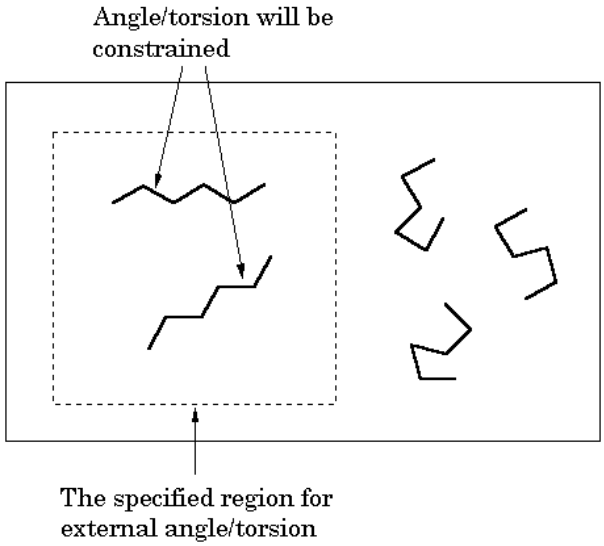

Figure 2.3: External angle/torsion potentials

- Homogeneous field

  Homogeneous external field $\mathbf{V}$ given by the 3-dimensional vector is applied to interaction sites.

$$U_{ext} = \mathbf{V} \cdot \mathbf{r} \tag{2.87}$$

$$\frac{dU_{ext}}{d\mathbf{r}} = \mathbf{V} \tag{2.88}$$

- External velocity field

  The external velocity field, which is obtained from Muffin_phaseseparation, is applied to the atoms by the SLLOD algorithm [**?**] as shown in eq.2.89-2.90.

$$\dot{\mathbf{q}}_i = \frac{\mathbf{p}_i}{m} + \mathbf{q}_i \cdot \nabla \mathbf{u} \tag{2.89}$$

$$\dot{\mathbf{p}}_i = \mathbf{F}_i - \mathbf{p}_i \cdot \nabla \mathbf{u}, \tag{2.90}$$

  $\mathbf{u}$: streaming velocity.

  Note that the boundary conditions for **COGNAC** must correspond to those of Muffin_phaseseparation.

- Tethered Force

  Specified Interaction_Site will be constrained around the initial position $\mathbf{r_0}$ with eq.2.91.

$$U_{ext} = K(\mathbf{r} - \mathbf{r_0})^2 \tag{2.91}$$

- User External Potential

  Users can use their own potential functions which are defined in the source code level.

2.6.6 Electrostatic interaction

The electrostatic interactions between charges or between dipoles are calculated. The following methods are implemented in **COGNAC**.

- Cutoff

  Simple cutoff method, which is used in the calculation of nonbonding interaction. Coulomb interaction between point charges is calculated by eq.2.92,

$$U_{electostatic}(\mathrm{r}_{ij}) = \frac{q_i q_j}{\epsilon r} \qquad r_{ij} < r_c \tag{2.92}$$

  q_{iorj}: point charge of atom i or j, ϵ: Dielectric constant, r_c: Cutoff distance

- Cutoff with Debye function

  Coulomb interaction between point charges is calculated with dumping function as is shown in eq.2.93,

$$U_{electostatic}(\mathrm{r}_{ij}) = \frac{q_i q_j}{\epsilon r} exp(-\kappa r) \qquad r_{ij} < r_c \tag{2.93}$$

  q_{iorj}: point charge of atom i or j, ϵ: Dielectric constant, κ: Inverse of Debye length r_c: Cutoff distance

- Reaction field[**?**, **?**, **?**]

  When calculating the electrostatic interaction between dipoles or between point charges, the long range contribution arising from shorter cutoff of the interactions is corrected by the reaction field of a dielectric continuum.

- Ewald[**?**]

  The Coulomb interaction between point charges can be calculated by the Ewald sum method.

  Automatic setting of Ewald parameters
  When automatic setting of paramters for Ewald raciprocal space, each paramter is set by the followng equations,

$$\alpha = \sqrt{\pi} \times \left(\frac{5.5N}{V^2}\right)^{\frac{1}{6}} \tag{2.94}$$

$$n_{n,k,l} = \frac{\left(2\alpha\sqrt{11.5} \times L_{a,b,c}\right)}{2\pi} \tag{2.95}$$

  N : Number of atoms, V : Volume, $L_{a,b,c}$: Length of unit cell of each vector

- Field electrostatic[**?**]

  This method enables DPD simulation with electrostatic interaction. The charge on the atoms will be distributed to the regular lattice in a space, and electric field will be obtained by solving the Poisson's equation. The the detail of the algorithm in the reference.

- Particle-Particle Particle-Mesh(PPPM) method[**?**, **?**]

  The Coulomb interaction between point charges can be calculated by the PPPM method.

2.7 Initial structure generation

The functions to generate the initial coordinates of the given molecular architecture for the MD simulation are explained below.

2.7.1 Initial coordinates generation

- Random : generation of amorphous structures

  The end atom of chain is placed at random position and the chain is grown based on the equilibrium bond length defined by bond potential. There is an option which fixes the bending angle to the equilibrium angle defined by angle potential. And an option that controls the discrete states of torsion angle, i.e. two states of trans/cis or three states of trans/gauche, by specifying the temperature and the energy difference between the conformations is also available.
  Since the excluded volume effect is not taken into consideration at generating the initial structure, it is necessary to perform the structure relaxation described below.

- Helix : generation of helical structures

  The end atom of a chain is placed on the specified point on a lattice, and the helical structure is generated based on the specified array of the torsion angle. The lattice type can be selected from the simple lattice, the face-centered lattice and the body-centered lattice. And the arbitrary position in the lattice also can be specified to place the end atom of a chain. When there is a chiral center in the main chain, the specification of stereo structure, i.e. meso and racemic, is possible.

- Crystal : generation of crystalline structures

  Lattice constants, symmetrical operation, fractional coordinate, etc. are read from the UDF file, and arbitrary crystalline structures can be generated.

- Lamella : generation of semi-crystalline lamella structures

  Initial structures of semi-crystalline lamella are generated, so that the chain configuration of the amorphous region between the crystalline lamella reproduces the configuration predicted by the mean field theory.
  The details of the algorithm are explained in the following section.

- Generation of multiphase structures obtained by **SUSHI**

  COGNAC has a function which generates the initial atomic coordinates in the system reproducing the morphology of block copolymers and polymer blends calculated by **SUSHI** as an extension of amorphous structure generation.
  The details of algorithm are explained in the following section.

Density biased Monte Carlo method

In **COGNAC**, a new method was developed[**?**] to generate initial structures of chains from the spatial distribution of segments obtained by **SUSHI**. The method is explained below.

The SCF calculation can give the spatial distribution of segments in the phase separated state. Let $\phi_n(\boldsymbol{r})$ be the local volume fraction of segment n, $\phi_n(\boldsymbol{r})$ being the probability that a point $\boldsymbol{r}$ in space is occupied by the segment n of a chain which consists of N segments ($n = 1, 2, 3...N$). In numerical calculation, $\phi_n(\boldsymbol{r})$ is given as $\phi_n(i, j, k)$, where (i, j, k) is the index of the lattice point and $\phi_n(\boldsymbol{r})$ is calculated by the interpolation from the values of $\phi_n(i, j, k)$ of the nearby lattice points.
To generate the initial chain configuration, we grow chains using the following Monte Carlo procedure.

1. Placement of the first atom in space with probability $\phi_1(\boldsymbol{r})$

   We choose a point $\boldsymbol{r}$ randomly in the simulation cell and generate a uniform random number r in a range from 0 to 1. If $r \leq \phi_1(\boldsymbol{r})$, we accept the point and proceed to step (2). If $r > \phi_1(\boldsymbol{r})$, we try another point. We repeat this procedure until the generated position is accepted.

2. Chain growth

   Figure 2.4 illustrates the chain growth method. Suppose that the $(n-1)$-th atom is placed at $\boldsymbol{r}'$, then a point $\boldsymbol{r}$ is chosen on the sphere $|\boldsymbol{r} - \boldsymbol{r}'| = \ell$ (ℓ being the equilibrium length of bond between atoms $n-1$ and n). The chosen point is accepted with probability $\phi_n(\boldsymbol{r})$ by the rule described above.

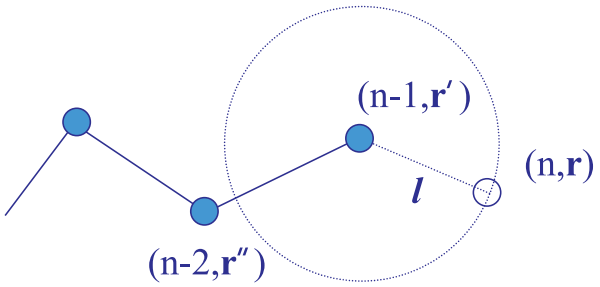

Figure 2.4: Density biased Monte Carlo

Lamella generator

To generate the initial structures for the semi-crystalline lamella systems, a new method was developed. In the method, the chain length and the conformation of interlamella amorphous region agree with those predicted by the mean field theory.
The outline of the method is described below.

The strategy for generating initial structures of semi-crystalline lamella

1. The length of long period of the semi-crystalline lamella and the length of crystalline phase are specified.
2. The densities of the crystalline and amorphous phase are specified.
3. The end of a chain is placed at random position in an amorphous phase.
4. Staring from one end of a chain, the chain is grown as a random walk manner until the front of the chain goes into a crystalline phase.
5. If the front of the chain goes into a crystalline phase, the helical structure of the crystal is grown in the crystalline phase until the front of the chain reaches an amorphous phase.

6. If the front of the chain goes into an amorphous phase, the atomic coordinates of in the amorphous phase will be generated based on a distribution of chain length and the loop/bridge conformation obtained by the mean field theory.
7. Step 5-6 is repeated until the chain grows to the end. In this step, if the chain end is placed in a crystalline phase, the step is terminated and the process is repeated from step 3 until the chain end is placed in an amorphous phase.
8. Step 3-7 are repeated by the number of molecules.
9. When the density of a crystalline phase is differ from the setting density within a certain criteria (the default is 1%), the generation is repeated from the beginning.
10. If the initial coordinates are generated, the structure relaxation will be performed in introducing the excluded volume effect.

The details for the step 6, which is newly developed for **COGNAC**, are explained below. First, the density of the whole amorphous chains d_a, which is a sum of the loop and bridge chains obtained from the mean field calculation, is given by

$$d_a = \frac{\sum_m \{n_l(m) + n_b(m)\} \times m}{L_a}, \tag{2.96}$$

where, $n_l(m)$ and $n_b(m)$ are the probabilities of the loop and the bridge components of the chain length m obtained by the mean field calculation, respectively. L_a is the thickness of an amorphous phase. If the free chains, which is not involved in the crystalline phase, and the tail chains are ignored, d_a normalized by the crystalline density must be 1.0. In the case of the MD simulation, the density of the crystalline phase and the amorphous phase can be set independently. Therefore, the probability p_a, where one chain goes into amorphous phase from crystalline phase, and the probability p_f, where one chain does not go into the amorphous phase but returns to crystalline phase (tight folding), are given by eqs.2.97 and 2.98, respectively, from a given ratio of the amorphous density d_a and crystalline density d_c, $r_d(= d_a/d_c)$.

$$p_a = r_d \sum_m \{n_l(m) + n_b(m)\} \tag{2.97}$$

$$p_f = 1 - p_a \tag{2.98}$$

The summation in eq.2.97 represents the number of chains in the amorphous phase per one chain in the crystalline phase. Thus, if the tail part of chains are ignored, the term becomes p_a in the case of $r_d = 1$. However, in this method, a tight folding chain is supposed to be a loop chain of length 1. Thus, the normalized density $d'_a(= r_d)$ of the amorphous phase is given by

$$d'_a = \frac{\left[p_a \sum_m \{n_l(m) + n_b(m)\} \times m\right] + \left[1 - p_a \sum_m \{n_l(m) + n_b(m)\}\right]}{L_a}, \tag{2.99}$$

where the first term in the right-hand side comes from the amorphous phase based on the mean field theory and the second term comes from the tight folding. The p_a and p_f are derived from eq.2.99 and given by

$$p_a = \frac{r_d L_a - 1}{\sum_m \{n_l(m) + n_b(m)\} \times m - \sum_m \{n_l(m) + n_b(m)\}} \tag{2.100}$$

$$p_f = 1 - p_a. \tag{2.101}$$

Actually, when a chain is extended to the end of a crystalline phase in the process of the chain generation, the molecular structure (loop chain/bridge chain, and degree of polymerization m) to be generated as an amorphous component of the chain is chosen by the Monte Carlo procedure based on the $n_l(m)$ and $n_b(m)$ obtained by the mean field theory. However, in order to take a tight folding into consideration, the p_f obtained by eqs.2.100 and 2.101 is added to $n_l(1)$ obtained by the mean field theory and the summation is used for probability distributions. The coordinate of an amorphous part of length m is generated at random, and the trial is repeated until the end segment reach proper position for loop/bridge conformation.

2.7.2 Structure relaxation

The energetically relaxed structure can be created from the generated initial coordinates.
The structure relaxation can be done by either dynamics simulation or energy minimization.

Structure relaxation by dynamics simulation

If the setup potential functions are applied to the initial structure, very large forces will act on crowded atoms due to the excluded volume effect, and the simulation process will break down. Therefore, in **COGNAC**, the excluded volume effect is introduced gradually by scaling the forces acting on atoms. The maximum force allowed to act on atoms is given and if the forces become larger than the maximum force, the force is scaled to that value. Then the maximum force for scaling is increased gradually, if the averaged force and the maximum force acting on each atoms decrease in the relaxation process, and if the forces acting on all atoms become smaller than the initial maximum force for scaling, the relaxation will be completed.

Structure relaxation by energy minimization

The energy minimization can be performed by the combination of the steepest descent method and the conjugate gradient method to obtain energetically relaxed structures.

2.8 Boundary conditions

The following boundary conditions are available in **COGNAC**.

- Periodic boundary conditions
- Lees-Edwards boundary conditions [**?**]

  The boundary conditions for shear flow are supported.
- Staggered reflective boundary conditions [**?**]

  This is a modified reflective boundary condition for modeling the interface of polymer blends.

When the periodic boundary conditions are applied to the system, chemical bonds based on the periodic boundary conditions can be calculated in **COGNAC**; i.e. bond lengths, bending angles and torsion angles are determined by the minimum image distances of atoms for calculating potential energy. The function is useful for modeling of infinite chains and chemical reactions for examples.

Staggered reflective boundary conditions

When studying the interfacial structure of polymer blends, the determination of the boundary conditions at the bulk region is a problem. Conventionally, solid walls or periodic boundary conditions are imposed at the edge or the middle of the bulk region.
When solid walls are placed at both ends of the unit cell, the bulk region must be taken to be large enough to reduce the effect of the walls. This poses a serious problem in the case of polymers since the walls affect the dynamics of molecules in the range of more than the radius of gyration [**?**].
When the periodic boundary conditions are imposed, two interfaces must be created. Thus, the unit cell size must be two times larger than that is actually needed.
To overcome the problem, we introduced a new boundary condition for modeling polymer interfaces which we call the staggered reflective boundary condition(SRBC). This is implement in **COGNAC**. Figure 2.5 illustrates the staggered reflective boundary condition.

This is a modification of simple reflective boundary conditions. In the case of simple reflective boundary conditions, the image polymer created by the boundary conditions tends to overlap the original polymer in the simulation box, and gives a very large force due to the excluded volume interaction. To overcome the problem, we modified the reflective boundary condition, and created image polymers with a half distance of the unit cell length shifted parallel to the boundary plane as it is shown in Figure 2.5(b). The shift reduces the chance of the overlap of atoms. Though the method of SRBC will work only when the bulk region at the boundary plane is homogeneous, the structure and dynamics of molecules at the boundary are practically equivalent to those of the bulk region simulated with the periodic boundary conditions. Thus,

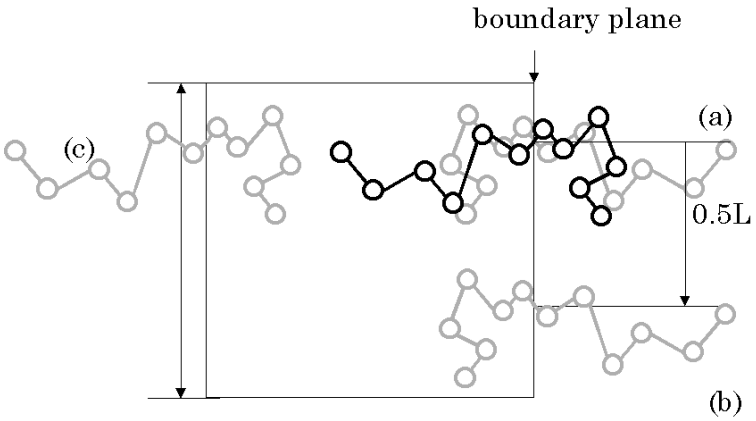

Figure 2.5: Staggered reflective boundary condition: Solid molecule is in real position and shaded molecules are the images of (a)Simple reflective boundary condition, (b)Staggered reflective boundary condition, (c)Periodic boundary condition

when studying the interfacial structures using the SRBC, the unit cell length can be half of that needed in the case of periodic boundary conditions, and the computational time can be reduced.

2.9 Chemical reaction

COGNAC has a function to deal with simple chemical reaction. The bonds connecting the atoms can be created or deleted during MD simulation according to a certain rule. Furthermore, atom type can be changed during MD simulation. The following is the scheme implemented at present.

- Bond creation
  When the distance of two reactive atoms becomes closer than the threshold distance, a new bond is created between the atoms with given probability, and the corresponding bond stretching potential is applied, while the non-bonding interaction between the atoms is removed. Angle bending and torsion angle potentials can be added too.
  Two types of reaction are implemented in current version. One is simple reaction, i.e. $A+B \rightarrow A-B$, another is mimic for radical or ion polymerization, i.e $R+M \rightarrow P-R$, where active end reacts with monomer, then active end becomes inactive unit and monomer becomes active end.
  The evaluation of bond creation is executed in a specified interval of the MD time steps. The reactivity can be basically controlled by the probability of reaction. The threshold distance and the interval of the judgment affect reactivity too.
  This algorithm of bond creation is suitable for the bead-spring type models in which the equilibrium bond length r_0 and the diameter of the Lennard-Jones potential σ are almost the same, since unconnected atoms can be close each other to the stable bond length.

- Atom type exchange
  Atom type can be changed with given probability when the atom with specified atom type moved to specified spacial area. This function is effective for the simulation of pseudo evaporation in which solvent atoms moved to gas phase and change to void atom.

- Bond scission
  When the length of the specified bond exceeds the threshold distance, The bond will be deleted and the non-bonding interaction between the atoms, which were at both ends of the deleted bond, is applied, when the criterias are satisfied. The criteria is the followings:

1. Bond length
   When the length of the specified bond exceeds the threshold distance, the bond will be deleted.
2. Region
   When a position of atom, which is either end of the bond is located in the specified region, the bond will be deleted.

This evaluation is done at every time step of the MD simulation. To prevent a discontinuous change of the force acting on atoms at scission, a proper bond potential function must be selected in which the force of the bond potential decreases smoothly at the threshold distance, such as the Morse type potential.

When the chemical reaction takes place, the Hamiltonian of the system is generally not conserved. Therefore, if one wants to do the simulation at constant temperature, one has to use some non-Hamiltonian method such as the velocity scaling and the loose coupling methods.
The above bond creation/atom exchange/scission algorithm does not reflect the actual chemical reaction, e.g. catalysis and the condensation reaction, but it will be useful for studying various reactive processes of polymer chains such as polymerization, gelation and crosslinking.

2.10 Methods for calculating density distributions

Some functions to calculate the density distributions at regular lattice points from the atomic coordinates are implemented in **COGNAC**, in order to compare the calculation results of the density distributions with the results of the continuous model such as **SUSHI**. The methods are explained below. The extrapolation method described in method 2 is implemented in **COGNAC** code, while the analysis of 1D density distributions by using the Python script is done by method 1 (refer to chapter7).

Method 1 : simple domain division

A unit cell is divided into partial cells as shown in Fig.2.6. Then the density of each partial cell is calculated by counting the number of atoms which exist in the partial cell, then divided by the volume of the partial cell.
If the number of partial cells is small enough compared with the number of atoms, the density distribution can be calculated by this simple method within a sufficient accuracy.

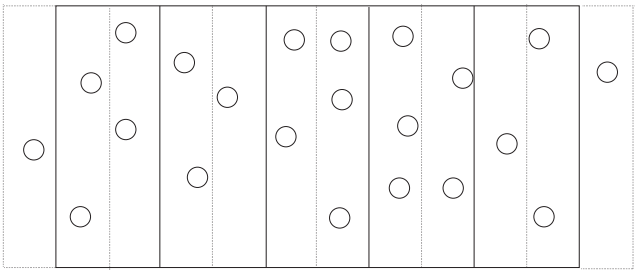

Figure 2.6: Density distribution by a simple domain division

Method 2 : Extrapolation method

If a unit cell is divided into three dimensions and the number of partial cells becomes the same order as the number of total atoms, the density distribution cannot be calculated by the simple domain division within

a sufficient accuracy. Then, the density distributions on lattice points are calculated by taking the volume of atoms into consideration. The details of this method are explained in Figure2.7. The Figure shows the example of 1D case. In the Figure, the lattice points are located at $-1.0, 0.0, 1.0, 2.0$ and atom is located at the position of 0.4. As shown in Fig.2.7(a), when the effective radius of atom is set at 1.0 as shown by the triangle, the existence probabilities of the atom on 2 lattice points (0.0) and (1.0) are given. When the effective radius of atom is set at 2.0, the existence probability of the atom is set at the 4 lattice points $(-1.0, 0.0, 1.0)$ and (2.0) as it is shown in Fig.2.7(b). Generally, when an atom is located at the position of z_{atom} and the effective radius is set at r, the existence probability of the atom at position z $\phi(z)$ is given by

$$\begin{aligned}
\phi(z) &= 0, & z \leq z_{atom} - r \\
\phi(z) &= 1.0 - \frac{z_{atom} - z}{r}, & z_{atom} - r < z \leq z_{atom} \\
\phi(z) &= 1.0 - \frac{z - z_{atom}}{r}, & z_{atom} < z \leq z_{atom} + r \\
\phi(z) &= 0, & z > z_{atom} + r.
\end{aligned} \tag{2.102}$$

In practice, the existence probabilities of the atom on lattice points are normalized so that $\sum_z \phi(z)$ becomes 1.0. The existence probabilities of all the target atoms are summed, and the densities are calculated by dividing this probabilities by partial cell volume.

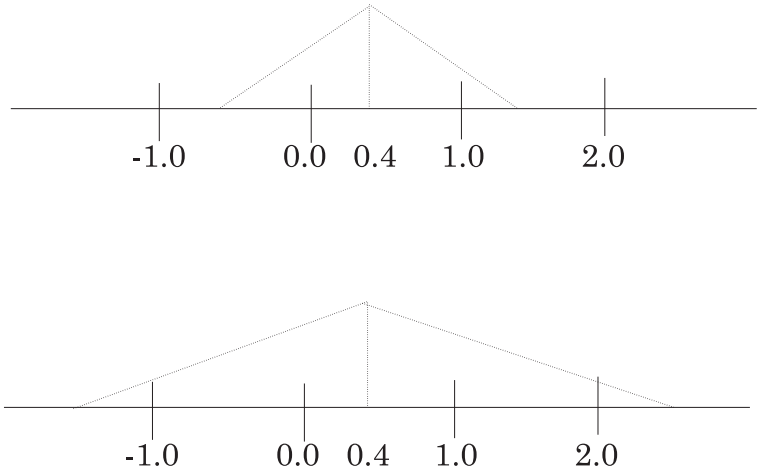

Figure 2.7: Density distribution by extrapolation method, (a) r=1.0, (b) r=2.0

In the case of three dimensions, existence probability at $\mathbf{r}(x, y, z)$ is given by

$$\phi(\mathbf{r}) = \phi(x)\phi(y)\phi(z) \tag{2.103}$$

2.11 On the fly calculation of autocorrelation function

Autocorrelation functions of various data such as position, velocity, bond vector, stress and energy are popular analysis method of molecular dynamics. Normalized autocorrelation function of data A, $C(t)$ is obtained by the euation 2.104, and many combination of data that is separated by time t are used.

$$C(t) = \left\langle \frac{A(t)A(0)}{A(0)A(0)} \right\rangle = \frac{1}{t_{max}} \sum_{t_0=1}^{t_{max}} \frac{A(t_0+t)A(t_0)}{A(t_0)A(t_0)} \tag{2.104}$$

Normally, autocorrelation functions are calulated from the trajectory and other data that are saved on output files after the simulation. Since, it needs huge disk space to save the data at every time step, only sparsed data are saved on the file.

COGNAC uses the algorithm of Magatti et al.[**?**] to calculate autocorrelation functions on the fly. The algorithm uses one-step data for the correlation of short interval, while the average of certain time steps are used for the correlation of long interval. Thus, the memory can be saved even the correlation of long time.

Version 8.3 provide the sutocorrelation function of stress. Furthermore, relaxation modulus $G(t)$ can be calculated by the Green-Kubo fomula, which is shown in eq.2.105.

$$G(t) = \frac{V}{k_B T} \langle \sigma(t)\sigma(0) \rangle \tag{2.105}$$

V:Volume, k_B:Boltzmann constant, T:Temperature, σ:Stress

Chapter 3

Starting COGNAC

This chapter explains how to start **COGNAC**. An example of the simulation of ABA triblock copolymers using a bead-spring model is explained. In the example, a procedure is explained to model molecules, set the calculation conditions, execute the **COGNAC** and analyze the results.
The files used here are placed in "sample/block" directory, if the directory path is not specified. When the directory path is specified, that path shows a relative path from "COGNAC" directory.

3.1 Preparation of the input data by Action SILK

This section explains how to define the topological data of 50 molecules of ABA triblock copolymer by **Action SILK**

1. Selection of the UDF file for the input of **Action SILK** ("potential_map.udf")

   First, refer to the UDF file for the input of **Action SILK** ("python/silk/sample/potential_map.udf"). Here, the 5th record (the record number is **4** and the record label is "BS_ABA_triblock") is used.

2. Load of the UDF file ("potential_map.udf")

   Start **GOURMET** and open UDF file "potential_map.udf" from the menu **File** → **Open...**. Since the data of the record number **0** is displayed on the window, move the slide bar under the **GOURMET** window right until the record label "BS_ABA_triblock" is displayed on a lower right window.

3. Definition of a set of molecules

   (a) Selection of the template command of **Action SILK**
   Click the folder like icon of **Set_of_Molecules** in the loaded "potential_map.udf" with right button of mouse, then some **Action SILK** command will be appeared as it is shown in Figure 3.1. (Since **Set_of_Molecules** is empty at initial state, the colored of icon is gray.) Select command **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...** from the appeared commands to model bead-spring type triblock copolymer.

   (b) Setting of **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...**
   When **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...** is selected in the command list shown in Figure 3.1, a new window will be appeared as it is shown in Figure 3.2. The parameters for modeling triblock copolymer are specified in this window.
   Here, 50 molecules of A20B40A20 triblock copolymer. In addition to the topological data of the molecule, atom type, bond potential and interaction site type will be specified.

   Description of parameters

   - **name** : Molecular name. Arbitrary name, e.g. **A20B40A20**。

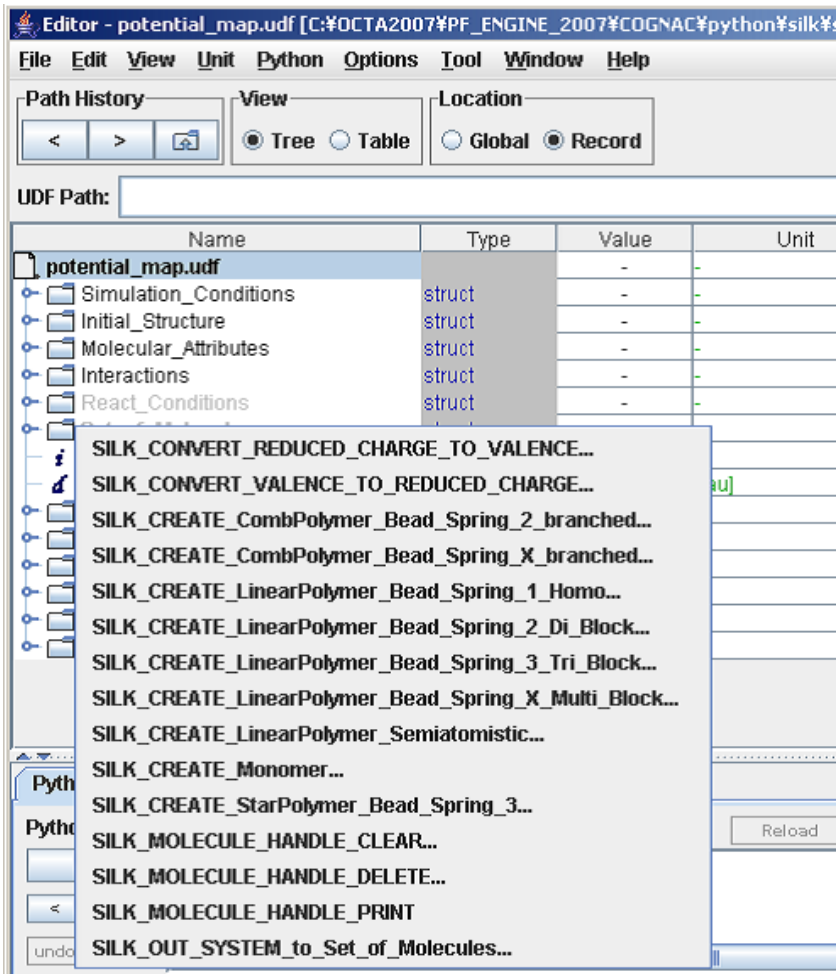

Figure 3.1: The command list of **Action SILK**

- **numMol** : Number of molecule. **50** in this example.

- **atom#_name** : Atom name of #-th block.
  A(#=0,2) and **B**(#=1) are set in this example.

- **atom#_num** : Length of #-th block.
  20(#=0,2) and **40**(#=1) set in this example.

- **atom#_type** : Atom type of #-th block.
  It must be selected from the items in **Molecular_Attribute** of "potential_map". **atom1**(#=0,2) and **atom2**(#=1) are selected in this example.

  (Notice) Atom_name can be specified arbitrarily. On the other hand, **Atom_type** is a type of atom, and the type must be included is **Molecular_Attribute**.

- **bond_type_in_#** : Bond potential type in #-th block.
  It must be selected from the items in **Molecular_Attribute** of "potential_map". **bond1**(#=0,2) and **bond2**(#=1) are selected in this example.

- **bond_type_#1_to_#2** : Bond potential type os the bond connecting #1-th and #2-th blocks.
  It must be selected from the items in **Molecular_Attribute** of "potential_map". **bond2**(#=0,2)

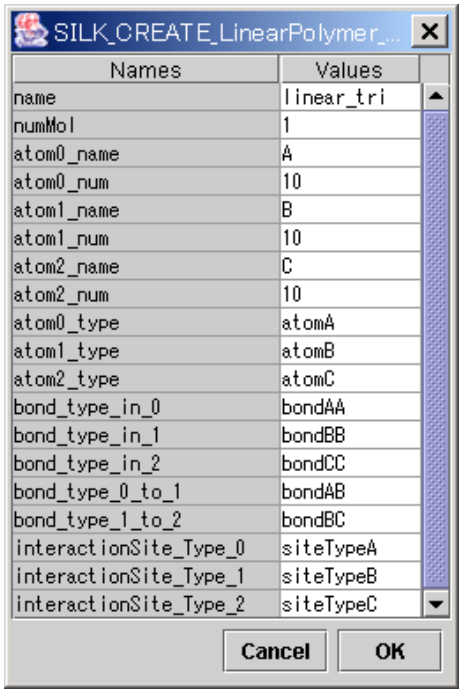

Figure 3.2: Initial state of the parameters for **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...**

is selected for both **0_to_1** and **1_to_2** in this example.

- **interactionSite_Type_#** : Interaction site type of #-th block.
  It must be selected from the items in **Molecular_Attribute** of "potential_map". **siteType1**(#=0,2) and **siteType2**(#=1) are selected in this example.

After all parameters are specified as it is shown in Figure 3.3, select **OK**. The information of created molecules are not put in **Set_of_Molecules** yet.

4. Output to **Set_of_Molecules**

   Since only one type of moelcule is defined in this example, put the created information of molecules to **Set_of_Molecules** now. If you will define multi type of molecules, repeat previous step before put the information in **Set_of_Molecules**.

   Click the folder like icon of **Set_of_Molecules** in the loaded "potential_map.udf" with right button of mouse again, then select **SILK_OUT_SYSTEM_to_Set_of_Molecules...** from the command list. When selecting **OK** in the appeared window with keeping parameter **Continue**, **Set_of_Molecules** is updated in the current record of "potential_map.udf". Notice that the data of **Set_of_Molecules** is only exist in the cache memory of **GOURMET**, and "potential_map.udf" file is not update until you save the UDF in **GOURMET**. It is better not to overwrite the template "potential_map.udf".

5. Create a new UDF file

   Create a new UDF file from the record of "potential_map.udf", where defined **Set_of_Molecules** and template parameter for calculation conditions.
   To do this, click the root icon of "potential_map.udf" with right button of mouse, then select **SILK_UDF_CREATE...** from the appeared command list. When specifying the file name in the appeared window and select **OK**, the input UDF file for **COGNAC** is created. The file name "A20B40A20_in.udf" is used for the following example.

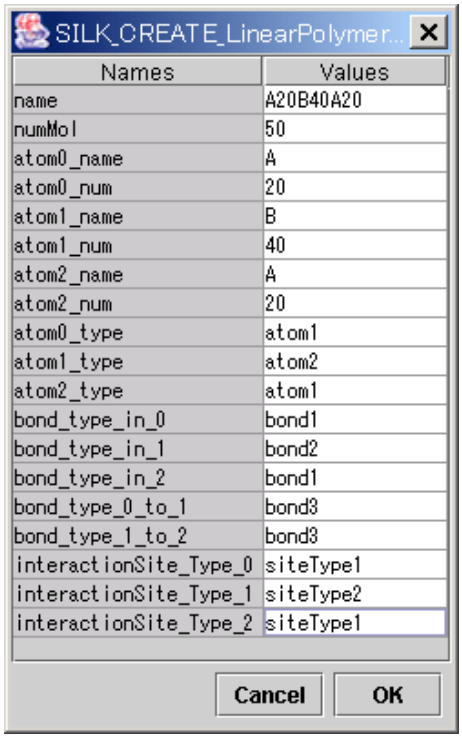

Figure 3.3: Setting parameters for **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...**

3.2 Edit of input data, and execution of COGNAC

Next, load the UDF file created by **Action SILK** into **GOURMET**, then after editting the content, execute **COGNAC**.

1. Load of the **COGNAC** input UDF created by **SILK**

   From **GOURMET Editor** window, select **File** → **Open...** and select "A20B40A20_in.udf" from the appeared file browser.
   If the input UDF is read and browse mode is selected by **Mode** option in a **Editor** window, the tree-like UDF structure will be displayed on the left-hand side of **GOURMET** window as shown in Figure 3.4.

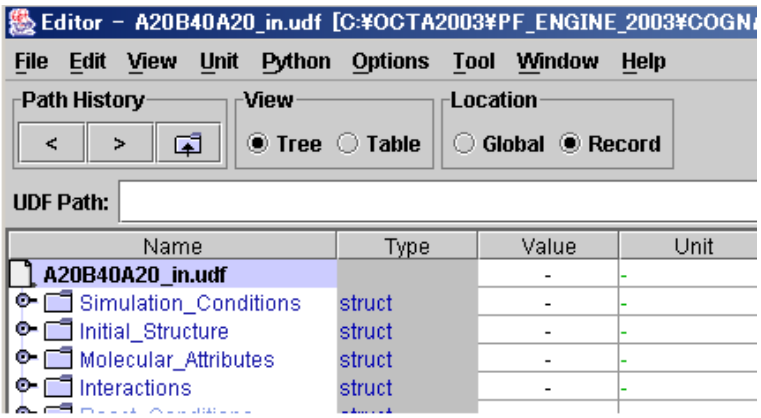

Figure 3.4: UDF structure in **GOURMET**

If **Table** is selected in **View** mode, the UDF structure is displayed as shown in Figure 3.5. In this manual, the usage of **Table** mode are explained if there is no notice.

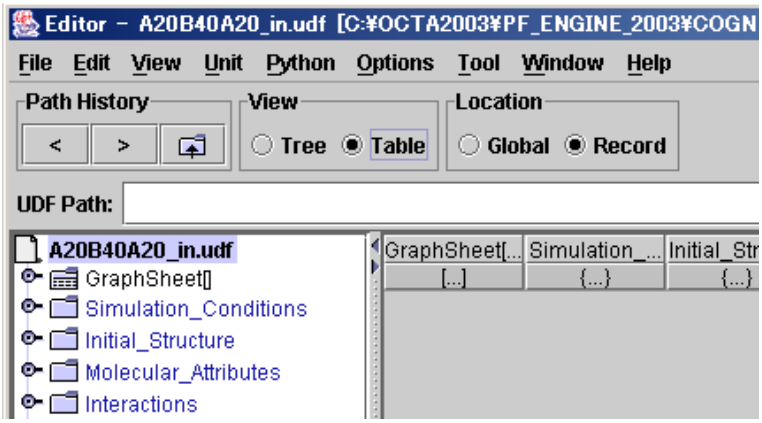

Figure 3.5: UDF structure (table mode) in **GOURMET**

The lower layer data is displayed by clicking folder-like icons of the tree structure currently displayed on left-hand side. For example, if **Simulation_Conditions** → **Dynamics_Conditions** → **Time** is selected, the input data about a time step will be displayed as shown in Figure 3.6.

2. Setting of the initial structure

   The information about initial structure is set up in the UDF path **Initial_Structure**. The main parameters of **Initial_Structure** are explained.

   Initial_Unit_Cell.Density ··· Density of the system specified as an initial condition

   Here, the reduced density **0.85** is specified.

   Generate_Method.Method ··· Method for initial structure generation

   When the **Value** cell is clicked with the left button of mouse, the select menu will be appeared. **Random** is selected in this example and the data below the UDF path **Generate_Method.Random** is referred.

3. Setting of the calculation conditions

   The calculation conditions are set up by editing the UDF path **Simulation_Conditions**. The setup parameters are as follows.

   (a) Dynamics condition

   The data in the UDF path **Simulation_Conditions.Dynamics_Conditions** is explained.

   Time.delta_T (setting value : **0.012**) ··· Time of each dynamics step

   Time.Total_Steps (setting value : **1000**) ··· Total time steps of the simulation

   Time.Output_Interval_Steps (setting value : **100**) ··· Interval of output

   Temperature.Temperature (setting value : **1.0**) ··· Target temperature

   Temperature.Interval_of_Scale_Temp (setting value : **0**) ··· Number of interval steps for velocity scaling

   * Velocity scaling is not performed when **Temperature.Interval_of_Scale_Temp** is set as **0**. Here, since temperature control is carried out by other technique, there is no necessity of the velocity scaling.

   (b) Solver

   The data in the UDF path **Simulation_Conditions.Solver** is explained.

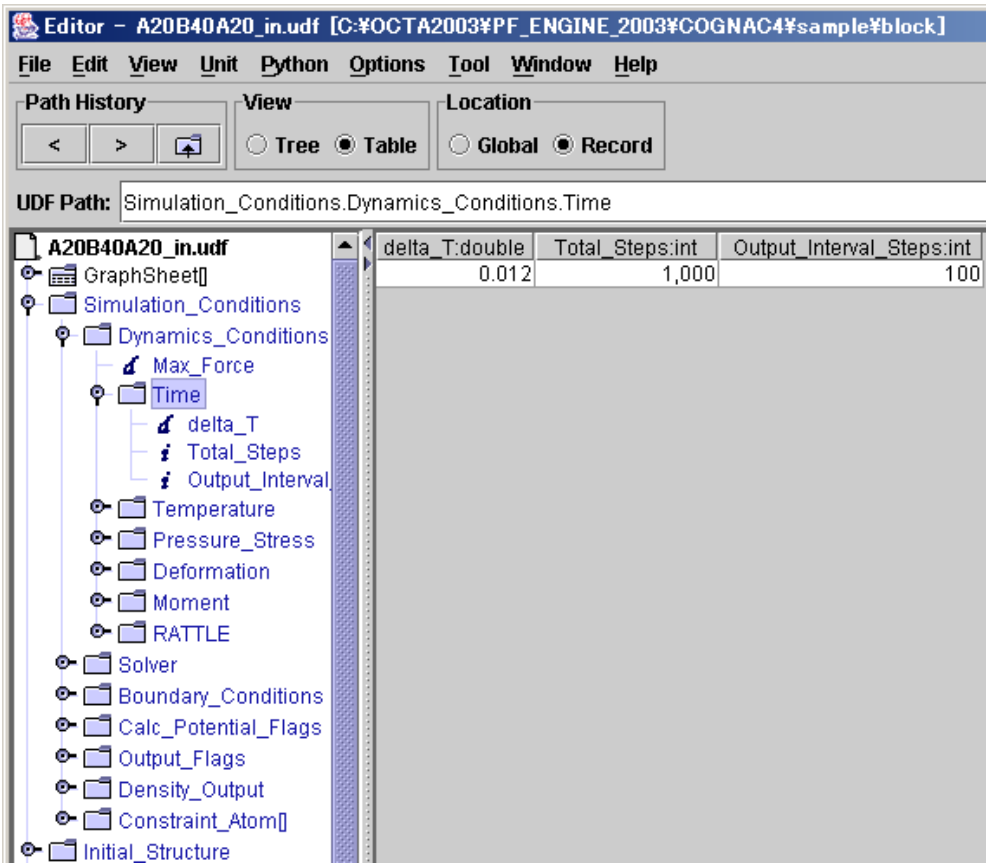

Figure 3.6: Constents of **Simulation_Conditions.Dynamics_Conditions.Time**

Solver_Type (setting value : **Dynamics**) ··· Selection of dynamics (**Dynamics**) or minimization (**Minimize**)

Here, **Dynamics** is selected.

Dynamics.Dynamics_Algorithm (setting value : **NVT_Kremer_Grest**) ··· Algorithm of dynamics

Here, the Langevin dynamics is selected.

Dynamics.NVT_Kremer_Grest.Friction (setting value : **0.5**) ··· Parameter required for **NVT_Kremer_Grest** algorithm

(c) Boundary conditions

The data in the UDF path **Simulation_Conditions.Boundary_Conditions** is explained.

Boundary_Conditions ··· Boundary condition in each axis of a unit cell

3D periodic boundary conditions are set by setting all of **a_axis**, **b_axis** and **c_axis** to **PERIODIC**.

(d) Flags for calculating potential terms

The data in the UDF path **Simulation_Conditions.Calc_Potential_Flags** is explained.

Bond (setting value : **1**)
Angle (setting value : **0**)
Torsion (setting value : **0**) ··· Calculation conditions of the bonding potentials

In the case of a bead-spring model, the potentials of angle bending and rotation of torsion are not calculated. Therefore, the values are set as **0**.

Non_Bonding (setting value : **1**) ··· Calculation conditions of the non-bonding interactions

Non_Bonding_1_4 (setting value : **0**) $\cdots$ Calculation conditions of the 1-4 non-bonding interactions. This is valid only when torsion potential is on

Non_Bonding_Intrachain (setting value : **0**) $\cdots$ Calculation conditions of non-bonding interaction between the beads in the same molecule. Normally this is set to **1**.

(e) Setting of the output data item
The data in the UDF path **Simulation_Conditions.Output_Flags** is explained.

Statics $\cdots$ Selection of output items, such as temperature and pressure. (Selected data are output to data and log file. However, all data are output to UDF file.)

Structure.Position (setting value : **1**)
Structure.Velocity (setting value : **1**)
Structure.Force (setting value : **0**) $\cdots$ Selection of the output of the position, velocity and force of each atoms

The position and velocity of atoms are output.

4. Save of the UDF

After each parameter is edited, the UDF file should be updated from the menu, **File** $\rightarrow$ **Save** in **GOURMET**.

5. Execution of **COGNAC**

(Note) Make sure the **Engine Manager** is running before run OCTA engines. If the **Engine Manager** is not running, select **StartEngineManager** from Start menu **OCTA8.3** in the case of Windows operating system.

By selecting **GOURMET Tool** $\rightarrow$ **Engine Run...** from the menu, **Engine Run** panel is opened. After setting the following parameters, **COGNAC** is started with the **Run** button. Other parameters in **Engine Run** panel can be kept as they are.

- **Engine**: $\cdots$ Open a file browser from the **[...]** button at the right end and choose a **COGNAC** executable file ("cognac92{.exe}"). Usually, the executable file is installed in **$(PF_ENGINE)/bin/$(PF_ENGINEARCH)**.
(**$PF_ENGINE** and **$PF_ENGINEARCH** are the environment variables, which specify the directory where the simulation programs of OCTA are installed, and the environment (linux, cygwin etc.) where a simulation program is performed, respectively. They must be set up beforehand. Refer to the "OCTA installation guide" for details.)

- **Working Dir**: $\cdots$ Specify arbitrary working directory, e.g. "C:\temp" in Windows operation systems, to execute **COGNAC**. Don't specify the directory where input UDF is located.

- **Output UDF**: $\cdots$ Specify as "A20B40A20_out.udf" as a output UDF with full path name.

- **Summary UDF**: $\cdots$ Specify arbitrary UDF name (e.g. "summary.udf" with full path name) for monitoring the calculation from **GOURMET Engine Control** panel.

It's recommended that to clear the check box of **Display Engine Control Window when Run Engine** in the **Engine Run Panel**, so that the log window remain after the engine run finished for checking error message.
When engine is started, **Engine Control** panel is appeared and some data such as time step, temperature and pressure will be displayed for monitoring the calculation. Since data is not updated during the relaxation process of initial structure, it will take some time before the data is updated.
After the simulation is completed, "A20B40A20_out.udf" (UDF output) and "A20B40A20_out.dat" (data output) are created.

3.3 Display and analysis of a calculation results

This section explains how to display and analysis the calculation results.

1. Display of molecules using **Action**

   Load calculation results into **GOURMET** and display molecular structure using **Action**.

   (a) Load of the output UDF
   From **GOURMET Editor** window, select **File** → **Open...** and choose "A20B40A20_out.udf" from the appeared file browser. Then move to arbitrary record to display the molecular structure.

   (b) Execution of **Action**
   By clicking the icon "A20B40A20_out.udf" in the tree structure with the right button of mouse, a list of registered **Action** commands will be appeared as shown in Figure 3.7.

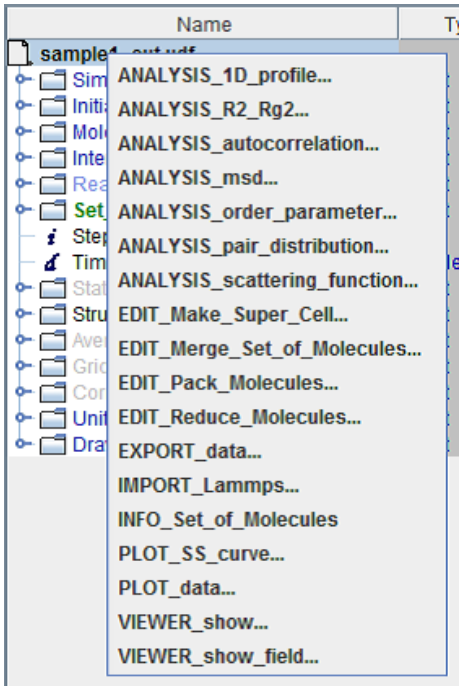

Figure 3.7: List of **Action** commands

When **VIEWER_show...** is selected in the list, a new window will be appeared as shown in Figure 3.8. With specifying parameters of **type**, **bc**(boundary_conditions), **color** in this window, various types of display style can be selected.

Description of parameters

- **type** ··· Drawing type
  It is selected from **line/ball-stick/rod/volume**.

- **bc** ··· Flag of whether to display molecules with applying boundary conditions.
  It is selected from **mol/atom/off**.

  mol : The image molecules, whose center of mass is in a unit cell, are displayed.
  atom : The image atoms in a unit cell are displayed.
  off : Original position of atoms are displayed.

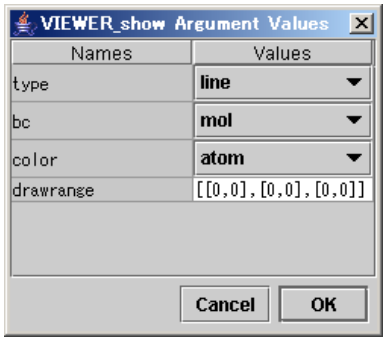

Figure 3.8: Window for setting parameter of **show_by_param...**

- **color** ··· Option of display color
  It is selected from **atom/bond/mol/molname**.

  atom : The color of atoms is selected according to the atom type.
  bond : The color of bonds is selected according to the bond potential.
  mol : The molecules are drawn by the different color for every molecules.
  molname : The color of molecules is selected according to the name of molecules.

- **drawrange** ··· Option of the range of display
  The image molecules can be displayed in addition to those in basic unit cell. The range of the display is specified in the list as follows,
  [[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
  The number of cell to display in a,b and c axis is specified in *amin,amax,bmin...*, e.g. if the variable is set [[-1,1],[-1,1],[-1,1]], image molecules in the unit cells which are shifted by minus one and plus one for each axis is display in addition to the base (central) unit cell.

An example of displaying molecular structure in **Viewer** window is shown in Figure 3.9.

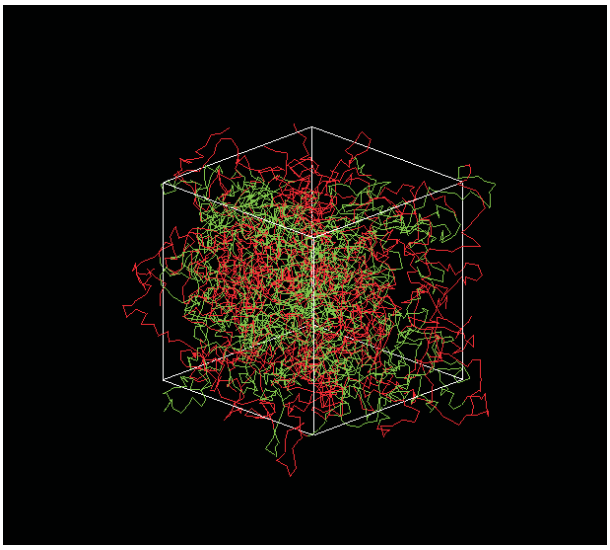

Figure 3.9: Displaying molecular structure

2. Analysis of radius of gyration

Calculation of the radius of gyration of molecules.

(a) Load of the output UDF
From **GOURMET Editor** window, select **File** → **Open ...** and choose "A20B40A20_out.udf" from the appeared file browser. It is unnecessary if the UDF file has already been loaded.

(b) Execute **Action**
Move to arbitrary record with the slide bar under the window for analysis. Next, select **ANALYSIS_R2_Rg2...** from the appearing **Action** commands by clicking the icon "A20B40A20_out.udf" in the tree structure with the right button of mouse. The parameter window will be appeared as shown in the Figure 3.10, then select **Rg** in **item** and **OK**.

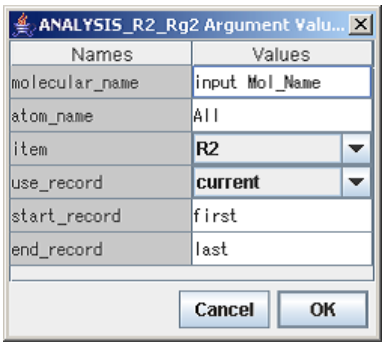

Figure 3.10: Parameter window of **ANALYSIS_R2_Rg2...**

The structure in the record will be analyzed, and the radius of gyration Rg^2 of each molecule and its components Rg_x^2, Rg_y^2 and Rg_z^2 will be calculated.

3. Analysis of pair distribution functions

Calculation of pair distribution functions between A and B atoms.

(a) Load of the output UDF
From **GOURMET Editor** window, select **File** → **Open...** and choose "A20B40A20_out.udf" from the appeared file browser. It is unnecessary if the UDF file has already been loaded.

(b) Execute **Action**
Move to arbitrary record with the slide bar under the window for analysis. Next, select **ANALYSIS_pair_distribution...** from the appearing **Action** commands by clicking the icon "A20B40A20_out.udf" in the tree structure with the right button of mouse. The parameter window will be appeared as shown in the Figure 3.11, then select **OK**.
The structure in the record will be analyzed, and the pair distribution function $g_{AB}(r)$ will be plotted as shown in Figure 3.12. See details of the parameter in section 7.5

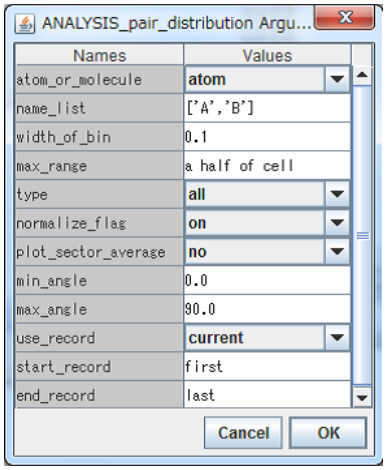

Figure 3.11: Parameter window of **ANALYSIS_pair_distribution...**

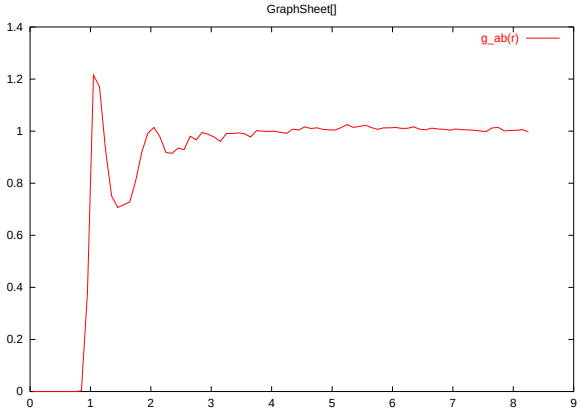

Figure 3.12: Plot of pair distribution

Chapter 4

Sample problems

4.1 Sample data list

Some samples of input/output UDF and scripts are collected in "sample" directory. The contents of samples are listed below.

```
sample -|--     blend      ----  blend_de01_in.udf
        |                    |--  blend100_silk_str.udf
        |                    |--  sushi_chiN200_uin.udf
        |                    |--  sushi_chiN200_uot.udf
        |                    |
        |                    |--  out ---- blend_de01_out.udf
        |                                  |-- blend_de01_out.dat
        |                                  |-- blend_de01.log
        |
        |--     block      ----  A20B40A20_in.udf
        |                    |--  A20B40A20_in_str.udf
        |                    |--  A20B40A20_sushi9_uin.udf
        |                    |--  A20B40A20_sushi9_uot.udf
        |                    |--  A20B40A20_zoom_in.udf
        |                    |--  density1D.py
        |                    |--  gr.py
        |                    |--  Rg.py
        |                    |
        |                    |--  out ---- A20B40A20_out.udf
        |                                  |-- A20B40A20_out.dat
        |                                  |-- A20B40A20.log
        |                                  |-- A20B40A20_zoom_out.udf
        |                                  |-- A20B40A20_zoom_out.dat
        |                                  |-- A20B40A20_zoom.log
        |
        |--   correlation  ----  n40_200_in.udf
        |                    |--  n40_200_rst.udf
        |                    |
        |                    |--  out ---- n40_200_out.udf
        |                                  |-- n40_200_out.dat
        |                                  |-- n40_200.log
        |
        |--     crystal    ----  pemodel20_crystal_in.udf
        |                    |--  cryst_xtl.udf
        |                    |
        |                    |--  out ---- pemodel20_crystal_out.udf
```

```
|                              |-- pemodel20_crystal_out.dat
|                              |-- pemodel20_crystal.log
|
|--    depletion   ----  depletion_in.udf
|                   |--  depletion_sushi9_uin.udf
|                   |--  depletion_sushi9_uot.udf
|                   |
|                   |--  out ---- depletion_out.udf
|                              |-- depletion_out.dat
|                              |-- depletion.log
|
|--     DPD        ---- A5B5_in.udf
|                   |
|                   |--  out ---- A5B5_out.udf
|                              |-- A5B5_out.dat
|                              |-- A5B5.log
|
|--  externalflow  ---- externalflow_in.udf
|                   |-- n10_650_eq.udf
|                   |-- Muffin_shear_out.udf
|                   |
|                   |--  out ---- externalflow_out.udf
|                              |-- externalflow_out.dat
|                              |-- externalflow.log
|
|--  GBLJpotential ----  GBLJpotential_in.udf
|                   |
|                   |--  out ---- GBLJpotential_out.udf
|                              |-- GBLJpotential_out.dat
|                              |-- GBLJpotential.log
|
|--   GBpotential  ----  GBpotential_in.udf
|                   |--  order.py
|                   |
|                   |--  out ---- GBpotential_out.udf
|                              |-- GBpotential_out.dat
|                              |-- GBpotential.log
|
|--      graft     ----  graft_in.udf
|                   |
|                   |--  out ---- graft_out.udf
|                              |-- graft_out.dat
|                              |-- graft.log
|
|-- infiniteChain  ---   inf  ----- infiniteChain_in.udf
|                   |
|                   |-- inf_PR ---- infiniteChainPR_in.udf
|                   |
|                   |--   out --|-- infiniteChainPR_out.udf
|                               |-- infiniteChainPR_out.dat
|                               |-- infiniteChainPR.log
|                               |-- infiniteChain_out.udf
|                               |-- infiniteChain_out.dat
|                               |-- infiniteChain.log
|
|--    lamella     ----  bs_lamella_in.udf
|                   |--  bs_lamella_in_str.udf
```

```
|                   |--  ua_lamella_in.udf
|                   |--  ua_lamella_in_str.udf
|                   |--  FileConvert.py
|                   |
|                   |--  out ---- bs_lamella_out.udf
|                              |-- bs_lamella_out.dat
|                              |-- bs_lamella.log
|                              |-- ua_lamella_out.udf
|                              |-- ua_lamella_out.dat
|                              |-- ua_lamella.log
|
|--    MDSCF       ----  a15b15_mdscf_in.udf
|                   |
|                   |--  out ---- a15b15_mdscf_out.udf
|                              |-- a15b15_mdscf_out.dat
|                              |-- a15b15_mdscf.log
|
|--    Minimize    ----  minimize_in.udf
|                   |
|                   |--  out ---- minimize_out.udf
|                              |-- minimize_out.dat
|                              |-- minimize.log
|
|-- molfile_sample ---- c14.mol   (File import のサンプル。付録 C 参照)
|
|--      NPT       ---- andersen  ---- andersen_in.udf
|                   |
|                   |-- berendsen ---- berendsen_in.udf
|                   |
|                   |--    prnh   ---- prnh_in.udf
|                   |
|                   |--    BC     ---- BC_in.udf
|                   |
|                   |-- rst_files ---- npt_rst.udf
|                   |
|                   |--    out    ---- andersen_out.udf
|                                       |-- andersen_out.dat
|                                       |-- andersen.log
|                                       |-- berendsen_out.udf
|                                       |-- berendsen_out.dat
|                                       |-- berendsen.log
|                                       |-- prnh_out.udf
|                                       |-- prnh_out.dat
|                                       |-- prnh.log
|                                       |-- BC_out.udf
|                                       |-- BC_out.dat
|                                       |-- BC.log
|
|---   pentaneNVT   ---  pentane50_in.udf
|                   |--  pentane50_rst.udf
|                   |--  MakeGraphSheet.py
|                   |
|                   |--  out ---- pentane50_out.udf
|                              |-- pentane50_out.dat
|                              |-- pentane50.log
|
|--      peo       ----  peo_cutoff_in.udf
```

```
|                   |--  peo_ewald_in.udf
|                   |--  peo_pppm_in.udf
|                   |--  peo_rf_in.udf
|                   |--  peo_in_str.udf
|                   |--  peo_rst.udf
|                   |--  msd.py
|                   |
|                   |--  out ---- peo_cutoff_out.udf
|                              |-- peo_cutoff_out.dat
|                              |-- peo_cutoff.log
|                              |-- peo_ewald_out.udf
|                              |-- peo_ewald_out.dat
|                              |-- peo_ewald.log
|                              |-- peo_pppm_out.udf
|                              |-- peo_pppm_out.dat
|                              |-- peo_pppm.log
|                              |-- peo_rf_out.udf
|                              |-- peo_rf_out.dat
|                              |-- peo_rf.log
|
|--- polymerization----  polymerization_in.udf
|                   |
|                   |--  out ---- polymerization_out.udf
|                              |-- polymerization_out.dat
|                              |-- polymerization.log
|
|--     RATTLE     ----  rattle_1_in.udf
|                   |--  rattle_2_in.udf
|                   |--  rattle_3_in.udf
|                   |--  rattle_4_in.udf
|                   |--  rattle_5_in.udf
|                   |--  norattle_1_in.udf
|                   |--  norattle_2_in.udf
|                   |--  norattle_3_in.udf
|                   |
|                   |-- rst_files ---- rattle_rst.udf
|                   |
|                   |--   out  ---- rattle_1_out.udf
|                                  |-- rattle_1_out.dat
|                                  |-- rattle_1.log
|                                  |-- rattle_2_out.udf
|                                  |-- rattle_2_out.dat
|                                  |-- rattle_2.log
|                                  |-- rattle_3_out.udf
|                                  |-- rattle_3_out.dat
|                                  |-- rattle_3.log
|                                  |-- rattle_4_out.udf
|                                  |-- rattle_4_out.dat
|                                  |-- rattle_4.log
|                                  |-- rattle_5_out.udf
|                                  |-- rattle_5_out.dat
|                                  |-- rattle_5.log
|                                  |-- norattle_1_out.udf
|                                  |-- norattle_1_out.dat
|                                  |-- norattle_1.log
|                                  |-- norattle_2_out.udf
|                                  |-- norattle_2_out.dat
```

```
|                                |-- norattle_2.log
|                                |-- norattle_3_out.udf
|                                |-- norattle_3_out.dat
|                                |-- norattle_3.log
|
|--    reaction    ----  react3_in.udf
|                    |--  react3_in_str.udf
|                    |--  CountMol.py
|                    |
|                    |--  out ---- react3_out.udf
|                                |-- react3_out.dat
|                                |-- react3.log
|
|-- tablepotential ----  pentane50_table_in.udf
|                    |--  pentane50_rst.udf
|                    |--  bond_table.udf
|                    |--  angle_table.udf
|                    |--  torsion_table.udf
|                    |--  nb_table.udf
|                    |
|                    |--  out ---- pentane50_table_out.udf
|                                |-- pentane50_table_out.dat
|                                |-- pentane50_table.log
|
|-- tailCorrection ----  andersen_short_in.udf
|                    |--  npt_rst.udf
|                    |
|                    |--  out ---- andersen_short_out.udf
|                                |-- andersen_short_out.dat
|                                |-- andersen_short.log
|
|
|--      tpe       ----  tpe_in.udf
|                    |--  tpe_shear_in.udf
|                    |--  tpe_rst.udf
|                    |
|                    |--  out ---- tpe_out.udf
|                                |-- tpe_out.dat
|                                |-- tpe.log
|                                |-- tpe_shear_out.udf
|                                |-- tpe_shear_out.dat
|                                |-- tpe_shear.log
|
|--      tpe2      ----  a5b73a5_in.udf
|                    |--  a5b73a5_rst.udf
|                    |
|                    |--  out ---- a5b73a5_out.udf
|                                |-- a5b73a5_out.dat
|                                |-- a5b73a5.log
|
|--      wall      ----  wall35_in.udf
                     |--  wall35_atom_in.udf
                     |--  wall35_rst.udf
                     |
                     |--  out ---- wall35_out.udf
                                |-- wall35_out.dat
                                |-- wall35.log
```

```
|-- wall35_atom_out.udf
|-- wall35_atom_out.dat
|-- wall35_atom.log
```

The samples currently prepared are explained briefly. The files referred here are placed under the sample directory as shown in the above list.

4.1.1 Pentane

[Keyword] pentane molecule / united atom model / Nose-Hoover NVT

This is an example of the simulation of the melt state of alkane molecules.

- Calculation model
  Molecular model ··· United atom model [**?**]
  System size ··· 50 molecules of pentane
  Initial structure ··· Restart from an UDF file
- Calculation conditions
  Solver ··· Nose-Hoover NVT algorithm
  Boundary Conditions ··· 3D periodic boundary conditions
- Sample files
  Input UDF file ··· "pentaneNVT/pentane50_in.udf"
  Restart UDF file ··· "pentaneNVT/pentane50_rst.udf"
  Output UDF file ··· "pentaneNVT/out/pentane50_out.udf"
  Output data file ··· "pentaneNVT/out/pentane50_out.dat"
  Standard output file ··· "pentaneNVT/out/pentane50.log"
- Notes
  Refer to the section 4.2.

4.1.2 Poly(ethylene oxide)(PEO)

[Keyword] Electrostatic interaction / polyelectrolyte

This is an example of the simulation of a system which consists of Poly(ethylene oxide)(PEO) and LiI. Charge is set on every site of PEO (united atom model) and ions (Li^+, I^-), and the electrostatic interaction is calculated. Four methods to calculate the electrostatic interaction; Ewald summation, PPPM, reaction field and cutoff method are compared. A crown ether-like configuration is formed by the electrostatic interaction between Oxygen atoms (minus electric charge) of PEO and Lithium ion (positive electric charge).

- Calculation model
  Molecular model ··· United atom model
  System size ··· 5 molecules of PEO of the degree of polymerization $N = 12$, and 5 pairs of Li^+ and I^-
  Initial structure ··· Restart from an UDF file
- Calculation condition
  Solver ··· Nose-Hoover NVT algorithm
  Boundary condition ··· 3D periodic boundary condition
  Electrostatic interaction ··· Calculation by the Ewald summation "peo_ewald"
  Electrostatic interaction ··· Calculation by the PPPM method "peo_pppm"
  Electrostatic interaction ··· Calculation by the reaction field methiod "peo_rf"
  Electrostatic interaction ··· Calculation by the simple cut off "peo_cutoff"

- Sample files
  - Ewald summation
    Input UDF file ··· "peo/peo_ewald_in.udf"
    Output UDF file ··· "peo/out/peo_ewald_out.udf"
    Output data file ··· "peo/out/peo_ewald_out.dat"
    Standard output ··· "peo/out/peo_ewald.log"
  - PPPM
    Input UDF file ··· "peo/peo_pppm_in.udf"
    Output UDF file ··· "peo/out/peo_pppm_out.udf"
    Output data file ··· "peo/out/peo_pppm_out.dat"
    Standard output ··· "peo/out/peo_pppm.log"
  - Reaction field
    Input UDF file ··· "peo/peo_rf_in.udf"
    Output UDF file ··· "peo/out/peo_rf_out.udf"
    Output data file ··· "peo/out/peo_rf_out.dat"
    Standard output ··· "peo/out/peo_rf.log"
  - Cutoff
    Input UDF file ··· "peo/peo_cutoff_in.udf"
    Output UDF file ··· "peo/out/peoo_cutoff_out.udf"
    Output data file ··· "peo/out/peoo_cutoff_out.dat"
    Standard output ··· "peo/out/peoo_cutoff.log"
  - Common restart UDF file ··· "peo/peo_rst.udf"

- Notes
  Bond lengths are restrained by the RATTLE algorithm. Refer to the section 4.3.

4.1.3 Liquid crystal(1): Gay-Berne potential

[Keyword] Gay-Berne potential / two atoms site / dipole reaction field

This is an example of the simulation using the Gay-Berne interaction.
In **COGNAC**, interaction sites can be defined by two or more atoms. For example, to set up the potential of a non-spherical form such as the Gay-Berne potential, interaction site is defined by two atoms. The interactions are calculated from the position of those atoms.

- Calculation model
  Molecular model ··· Gay-Berne potential
  System size ··· 256 molecules of bi-atom molecule (two atoms define one Gay-Berne interaction site)
  Initial structure ··· Restart from an UDF file

- Calculation condition
  Solver ··· NVE ensemble
  Boundary conditions ··· 3D periodic boundary condition
  Electrostatic interaction ··· Dipole-dipole electrostatic interaction calculated by the reaction field method

- Sample files
  Input UDF file ··· "GBpotential/GBpotential_in.udf"
  Output UDF file ··· "GBpotential/out/GBpotential_out.udf"
  Output data file ··· "GBpotential/out/GBbpotential_out.dat"
  Standard output ··· "GBpotential/out/GBbpotential.log"

- Notes
  Bond lengths are restrained by the RATTLE algorithm. Therefore, bond potential is not calculated.

4.1.4 Liquid crystal(2): Gay-Berne – Lennard-Jones hybrid potential

This is an example of the model which combines the Gay–Berne potential and the Lennard-Jones potential.

- Calculation model
  System size ··· 256 molecules of 5CB(4-n-pentyl-4'-cyanobiphenyl)
  (The 4-methyl-4'-cyanobiphenyl portion is represented by an ellipsoid, and the part of a alkyl tail is represented by the chain of spheres)
  Molecular model ··· Combination of the Gay-Berne potential and the Lennard-Jones potential. Angle potential and torsion potential are applied in the part of alkyl tail.
  (The potential parameters are results of rough estimation and not accurate enough[**?**, **?**])
  Initial structure ··· Packing structure of FCC type lattice.

- Calculation condition
  Solver ··· Loose-coupling method for temperature control
  Boundary conditions ··· 3D periodic boundary conditions

- Sample files
  Input UDF file ··· "GBLJpotential/GBLJpotential_in.udf"
  Output UDF file ··· "GBLJpotential/out/GBLJpotential_out.udf"
  Output data file ··· "GBLJpotential/out/GBLJpotential_out.dat"
  Standard output ··· "GBLJpotential/out/GBLJpotential.log"

- Notes
  Bond lengths are restrained by the RATTLE algorithm. Therefore, bond potential is not calculated.

4.1.5 Thermoplastic elastomer

[Keyword] alternative block copolymer / uniaxial elongation/ shear flow

This is an example of the simulation of elongational and shear deformation of thermoplastic elastomer. The thermoplastic elastomer is modeled as an alternative block copolymer. The micro phase separation is induced by giving an attractive potential only between hard segments (A) and a repulsive potential to other pairs,

- Calculation model
  Molecular model ··· Bead-spring (harmonic bond) model
  System size ··· 48 molecules of (A3B21)×6 block-copolymer
  Initial structure ··· Restart from an UDF file

- Calculation condition
  Solver ··· NVE
  Boundary conditions ··· 3D periodic boundary conditions
  Deformation
  - Uniaxial elongation by unit cell deformation
  - Shear flow by SLLOD+Lees-Edwards boundary conditions

- Sample files
  - Uniaxial deformation
    Input UDF file ··· "tpe/tpe_in.udf"
    Output UDF file ··· "tpe/out/tpe_out.udf"
    Output data file ··· "tpe/out/tpe_out.dat"
    Standard output ··· "tpe/out/tpe.log"

  - Shear flow
    Input UDF file ··· "tpe/tpe_shear_in.udf"
    Output UDF file ··· "tpe/out/tpe_shear_out.udf"
    Output data file ··· "tpe/out/tpe_shear_out.dat"
    Standard output ··· "tpe/out/tpe_shear.log"
  - Common restart UDF file··· "tpe/tpe_rst.udf"

4.1.6 Thermo plastic elastomer (2)

[Keyword] ABA tri block copolymer / bcc spherical domain / Uniaxial elongation

This is an example of the simulation of elongational deformation of ABA triblock copolymer, which shows BCC sherical micorphase separated structure. The initial structure was prepared from the output of **SUSHI** by density biased Monte Carlo method. See the detail of the simulation in ref.[**?**]

- Calculation model
  Molecular model ··· Bead-spring (harmonic bond) model
  System size ··· 347 molecules of A5B73A5 triblock-copolymer
  Initial structure ··· Restart from an UDF file

- Calculation condition
  Solver ··· NVT Kremer Grest type
  Boundary conditions ··· 3D periodic boundary conditions
  Elongation ··· Uniaxial elongation by cell deformation

- Sample files Input UDF file ··· "tpe2/a5b73a5_in.udf"
  Restart UDF ··· "tpe2/a5b73a5_rst.udf" Output UDF file ··· "tpe2/out/a5b73a5_out.udf"
  Output data file ··· "tpe2/out/a5b73a5_out.dat"
  Standard output ··· "tpe2/out/a5b73a5.log"

4.1.7 Solid wall

[Keyword] bead-spring model / 2D periodic boundary conditions / LJ type flat wall potential / LJ atomic wall

This is an example of the simulation of polymer melts confined between the solid walls [**?**]. Two types of wall potential are included in the sample. See the reference[**?**] for the application.

- Calculation model
  Molecular model ··· Kremer-Grest type[**?**] bead-spring model
  System size ··· 100 molecules of chain of length, $N = 40$
  Unit cell ··· $11.71 \times 11.71 \times 35.0\sigma$
  Initial structure ··· Restart from an UDF file

- Calculation condition
  Solver ··· Langevin dynamics
  Boundary condition ··· 2D periodic boundary conditions to the x and y directions.
  External potential

  - LJ type flat wall potential[**?**] on the xy plane.
  - LJ atomistic wall potential on the xy plane.

- Sample files

  - Flat wall
    Input UDF file ··· "wall/wall35_in.udf"
    Output UDF file ··· "wall/out/wall35_out.udf"
    Output data file ··· "wall/out/wall35_out.dat"
    Standard output ··· "wall/out/wall35.log"
  - Atomistic wall
    Input UDF file ··· "wall/wall35_atom_in.udf"
    Output UDF file ··· "wall/out/wall35_atom_out.udf"
    Output data file ··· "wall/out/wall35_atom_out.dat"
    Standard output ··· "wall/out/wall35_atom.log"

  Common restart UDF file ··· "wall/wall35_rst.udf"

- Notes
  Since the temperature is controlled by thermal noise, the results will be changed depending on the run.

4.1.8 Graft chain

[Keyword] end grafted chain / bead-spring model / 2D periodic boundary conditions / LJ type flat wall potential

This is an example of the simulation of end grafted chains.

- Calculation model
  Molecular model ··· Kremer Grest type [**?**] bead-spring model
  System size ··· 100 molecules of chain of length $N = 40$
  Unit cell ··· $20.0 \times 20.0 \times 50.0\sigma$
  Initial structure ··· Random with fixing the position of end atoms.

- Calculation condition
  Solver ··· Langevin dynamics
  Boundary condition ··· 2D periodic boundary conditions to the x and y directions.
  External potential ··· LJ type flat wall potential [**?**] to the z direction.
  Constraint atom ··· Constraining atoms of an end of all chains on the xy plane at $z = 1.0$.

- Sample files
  Input UDF file ··· "graft/graft_in.udf"
  Output UDF file ··· "graft/out/graft_out.udf"
  Output data file ··· "graft/out/graft_out.dat"
  Standard output ··· "graft/out/graft.log"

- Notes
  The positions of end atoms can be specified in **Initial_Structure.Generate_Method.Random.Fix_End_Position[**
  when generating random position. Using this function, the end atoms of grafted chain will be fixed on
  the square lattice of the z=1.0 plane.

4.1.9 Periodic chain

[Keyword] polyethylene united atom model / periodic chain

This is an example of the simulation of the set of helical chains which has an infinite chain length with considering the periodic boundary conditions.

- Calculation model
  Molecular model ··· United atom model [**?**]
  System size ··· 20 molecules of infinite length chain which consists of 20 atoms

- Calculation condition
  Boundary conditions ··· 3D periodic boundary conditions and periodic chain
  Solver ··· NVE → Parrinello-Rahman NPT algorithm

- Sample files

  1. Initial structure generation & NVE Dynamics
     Input UDF file ··· "infiniteChain/inf/infiniteChain_in.udf"
     Output UDF file ··· "infiniteChain/out/infiniteChain_out.udf"
     Output data file ··· "infiniteChain/out/infiniteChain_out.dat"
     Standard output ··· "infiniteChain/out/infiniteChain.log"
  2. Parrinello-Rahman NPT dynamics
     Input UDF file ··· "infiniteChain/inf_PR/infiniteChainPR_in.udf"
     Restart UDF file ··· "infiniteChain/out/infiniteChain_out.udf"(what was created in the previous step)
     Output UDF file ··· "infiniteChain/out/infiniteChainPR_out.udf"
     Output data file ··· "infiniteChain/out/infiniteChainPR_out.dat"
     Standard output ··· "infiniteChain/out/infiniteChainPR.log"

- Notes
  The simulation is continued on step 2 with NPT ensemble using the result of step 1.

4.1.10 Table potential

This is an example of the simulation using table potential.

- Calculation model
  Molecular model ··· United atom model [**?**] (potential tables are created for each potential)
  System size ··· 50 molecules of pentane
  Initial structure ··· Restart from an UDF file

- Calculation condition
  Solver ··· Nose-Hoover NVT algorithm
  Boundary conditions ··· 3D periodic boundary conditions

- Sample files
  Input UDF file ··· "tablepotential/pentane50_table_in.udf"
  Restart UDF file ··· "tablepotential/pentane50_table_rst.udf"
  Output UDF file ··· "tablepotential/out/pentane50_table_out.udf"
  Output data file ··· "tablepotential/out/pentane50_table_out.dat"
  Standard output ··· "tablepotential/out/pentane50_table.log"
  Bond Potential Table UDF file ··· "tablepotential/bond_table.udf"
  Angle Potential Table UDF file ··· "tablepotential/angle_table.udf"
  Torsion Potential Table UDF file ··· "tablepotential/torsion_table.udf"
  Non-Bonding Potential Table UDF file ··· "tablepotential/nb_table.udf"

- Notes
  Refer to the section 4.5.

4.1.11 Generation of crystal structure

This is an example of modeling an initial structure of crystal.

- Calculation model
  Molecular model ··· United atom model [**?**]
  System size ··· 8 molecules of icosane
  Initial structure ··· Crystal structure[**?**] generated by the crystal generator.
- Calculation condition
  Solver ··· Nose-Hoover NVT algorithm
  Boundary conditions ··· 3D periodic boundary conditions
- Sample files
  Input UDF file ··· "crystal/pemodel20_crystal_in.udf"
  Output UDF file ··· "crystal/out/pemodel20_crystal_out.udf"
  output data file ··· "crystal/out/pemodel20_crystal_out.dat"
  Standard output ··· "crystal/out/pemodel20_crystal.log"
  Crystal structure data UDF file ··· "crystal/cryst_xtl.udf"

4.1.12 Comparison of NPT algorithms

[Keyword] NPT ensemble / united atom model / pentane

This is an example of comparison of various algorithms for NPT ensemble.

- Calculation model
  Molecular model ··· United atom model [**?**]
  System size ··· 50 molecules of pentane
  Initial structure ··· Restart from an UDF file
- Calculation condition
  Boundary conditions ··· 3D periodic boundary conditions
  Temperature control ··· Nose-Hoover algorithm is used when pressure/stress is controlled by the extended Hamiltonian methods, and the loose-coupling method is used when pressure/stress is controlled by the loose-coupling methods. Cutoff distance of the Lennard-Jones potential ··· 2.5σ with the tail correction for energy and pressure.
- Sample files
  - Common UDF file
    Restart UDF file ··· "NPT/rst_files/npt_rst.udf"
  - Andersen algorithm (extended Hamiltonian method with isotropic cell)
    Input UDF file ··· "NPT/andersen/andersen_in.udf"
    Output UDF file ··· "NPT/out/andersen_out.udf"
    Output data file ··· "NPT/out/andersen_out.dat"
    Standard output ··· "NPT/out/andersen.log"
  - Parrinello-Rahman algorithm (extended Hamiltonian method with anisotropic cell)
    Input UDF file ··· "NPT/prnh/prnh_in.udf"
    Output UDF file ··· "NPT/out/prnh_out.udf"
    Output data file ··· "NPT/out/prnh_out.dat"
    Standard output ··· "NPT/out/prnh.log"
  - Berendsen algorithm (loose-coupling method with isotropic cell)
    Input UDF file ··· "NPT/berendsen/berendsen_in.udf"
    Output UDF file ··· "NPT/out/berendsen_out.udf"
    Output data file ··· "NPT/out/berendsen_out.dat"
    Standard output ··· "NPT/out/berendsen.log"

  - Brown-Clarke algorithm (loose-coupling process with anisotropic cell)
    Input UDF file ··· "NPT/BC/BC_in.udf"
    Output UDF file ··· "NPT/out/BC_out.udf"
    Output data file ··· "NPT/out/BC_out.dat"
    Standard output ··· "NPT/out/BC.log"

- Notes
  Since the optimum value of parameters for the temperature control and the pressure control will be different by the system size, physical properties of material and other calculation conditions, we can't guarantee that the parameters in the sample are optimum. We comment on the selection of parameters used in this samples.

  - Nose-Hoover **Q** parameter
    It is set as 20 empirically. When unit conversion is carried out, it becomes $\mathrm{Q} = 40(kJ/mol)(ps)^2$. It is a little larger than that in the reference [**?**].
  - **Cell Mass**
    Cell mass is set as the same as the total mass of the system empirically ($\mathrm{M}_{total} = 250m$ for 50 molecules of pentane, where m is a unit of mass and equals to the mass of methylene unit). However, in the case of Parrinello-Rahman algorithm, the bigger value is better since deformation of cell angles become very large, especially in the case of liquid state such as this sample.
  - **tauT**
    It is set as $\tau_T = 0.1$ (converted to 0.2 ps). The value is taken from the reference [**?**].
  - Berendsen **tauP**
    It is set as $\beta = 4.9 \times 10^{-5}\mathrm{bar}^{-1}$, and $\tau_P = 0.1$ps from $\tau_P/\beta = 6.6$. The values are taken from the reference [**?**].
  - Brown-Clarke **tauP**
    It is set as $\tau_P = 66$ (converted to $5.28 \times 10^6 \mathrm{Pa \cdot s \cdot m}^{-1}$). The value is taken from the reference [**?**].

4.1.13 Test of tail correction

This is an example of the test of tail correction with changing the cut off distance of the Lennard-Jones potential.

- Calculation model
  Molecular model ··· United atom model [**?**]
  System size ··· 50 molecules of pentane
  Initial structure ··· Restart from an UDF file

- Calculation condition
  Solver ··· Andersen and Nose-Hoover algorithm for NPT ensemble
  Boundary conditions ··· 3D periodic boundary conditions
  Cutoff distance of the Lennard-Jones potential ··· 1.5σ with the tail correction for energy and pressure.

- Sample files
  Input UDF file ··· "tailCorrection/andersen_short_in.udf"
  Restart UDF file ··· "tailCorrection/npt_rst.udf"(same initial data as NPT test)
  Output UDF file ··· "tailCorrection/out/andersen_short_out.udf"
  Output data file ··· "tailCorrection/out/andersen_short_out.dat"
  Standard output ··· "tailCorrection/out/andersen_short.log"

- Notes
  The non-bonding energy and the density should agree with those of the sample "NPT/andersen"(cut off= 2.5σ), if the tail correction works.

4.1.14 Test of the RATTLE algorithm

This is an example of the test of the effect of time step with the RATTLE algorithm.

- Calculation model
  Molecular model ··· United atom model [**?**]
  System size ··· 50 molecules of pentane
  Initial structure ··· Restart from an UDF file

- Calculation condition
  Solver ··· NVE and no temperature control Boundary conditions ··· 3D periodic boundary conditions

- Sample files

  - Common UDF file
    Restart UDF file ··· "RATTLE/rattle_rst.udf"
  - RATTLE on
    Input UDF file ··· "RATTLE/rattle_*dt*_in.udf"
    Output UDF file ··· "RATTLE/out/rattle_*dt*_out.udf"
    Output data file ··· "RATTLE/out/rattle_*dt*_out.dat"
    Standard output ··· "RATTLE/out/rattle_*dt*.log"
  - RATTLE off
    Input UDF file ··· "RATTLE/norattle_*dt*_in.udf"
    Output UDF file ··· "RATTLE/out/norattle_*dt*_out.udf"
    Output data file ··· "RATTLE/out/norattle_*dt*_out.dat"
    Standard output ··· "RATTLE/out/norattle_*dt*.log"

  dt shows a time step. $\Delta t = dt \times 10^{-3}$, which is converted to $dt \times 2.0$(fs)

- Notes
  The results of NVE show the effect of time step for the conservation of the Hamiltonian.
  Notice that since the same restart data are used in the case of the RATTLE on and off, the bond potential is apart from an equilibrium state a little, in the case of the RATTLE off.

4.1.15 Minimization

[Keyword] pentane / united atom model / energy minimization

This is an example of energy minimization.

- Calculation model
  Molecular model ··· United atom model [**?**]
  System size ··· 50 molecules of pentane
  Initial structure ··· Random structure

- Calculation condition
  Solver ··· Minimize with **Cascade** option
  Boundary conditions ··· 3D periodic boundary conditions

- Sample files
  Input UDF file ··· "Minimize/minimize_in.udf"
  Output UDF file ··· "Minimize/out/minimize_out.udf"
  Output data file ··· "Minimize/out/minimize_out.dat"
  Standard output ··· "Minimize/out/minimize.log"

- Notes
  A random initial structure is generated and following energy minimization is performed by the combination of the steepest descent and the conjugate gradient method. Since the implementation of the conjugate gradient algorithm in **COGNAC** is not so intelligent, the convergence near the minimum point is not so good. (However, it is good enough for the initial structure generation for MD.)

4.1.16 Lamella structure of block copolymer: zoom in from SUSHI(1)

[Keyword] density biased Monte Carlo / density biased potential

This is an example of modeling lamella structures of block copolymers from the information obtained by **SUSHI**.

- Calculation model
  Molecular model $\cdots$ Kremer-Grest type [**?**] bead-spring model
  System size $\cdots$ 50 molecules of A20B40A20 triblock copolymer
  Unit cell $\cdots$ $10.98 \times 10.98 \times 39\sigma$
  Initial structure $\cdots$ Generation by the density biased Monte Carlo method

- Calculation condition
  Solver $\cdots$ Langevin dynamics
  Boundary conditions $\cdots$ 3D periodic boundary conditions
  External potential $\cdots$ Density biased potential

- Sample files
  Input UDF file $\cdots$ "block/A20B40A20_in.udf"
  Molecular structure data UDF file $\cdots$ "block/A20B40A20_in_str.udf"
  Output UDF file $\cdots$ "block/out/A20B40A20_out.udf"
  Output data file $\cdots$ "block/out/A20B40A20_out.dat"
  Standard output $\cdots$ "block/out/A20B40A20.log"
  SUSHI output UDF file $\cdots$ "block/A20B40A20_sushi_out.udf"

- Notes
  Refer to the sections 2.7.1,2.6.5 and 4.4

4.1.17 Interfacial structure of polymer blend: zoom in from SUSHI(2)

[Keyword] density biased Monte Carlo / staggered reflective boundary condition

This is an example of modeling the interfacial structures of polymer blend from the information obtained by **SUSHI**.

- Calculation model
  Molecular model $\cdots$ Kremer-Grest type [**?**] bead-spring model
  System size $\cdots$ Blend of 40 molecules of A100 homopolymers and 40 molecules of B100 homopolymers
  Unit cell $\cdots$ $14.00 \times 14.00 \times 48.00\sigma$
  Initial structure $\cdots$ Generation by the density biased Monte Carlo method

- Calculation condition
  Solver $\cdots$ Langevin dynamics
  Boundary condition $\cdots$ Staggered reflective boundary conditions (z direction)

- Sample files
  Input UDF file ··· "blend/blend_de01_in.udf"
  Molecular structure data UDF file ··· "blend/blend100_silk_str.udf"
  Output UDF file ··· "blend/out/blend_de01_out.udf"
  Output data file ··· "blend/out/blend_de01_out.dat"
  Standard output ··· "blend/out/blend_de01.log"
  SUSHI output UDF file ··· "blend/sushi_chiN200_out.udf"

- Notes
  Refer to the sections 2.7.1 and 2.8

4.1.18 Depletion : zoom in from SUSHI(3)

[Keyword] density biased Monte Carlo / staggered reflective boundary condition / solid wall

This is an example of modeling the the blend of polymer chain and low molecular weight molecules near a solid wall from the information obtained by **SUSHI**.

- Calculation model
  Molecular model ··· Kremer-Grest type [**?**] bead-spring model
  System size ··· 10 polymer chain of the length 100 and 100 molecules of single atom
  Unit cell ··· $6.36 \times 6.36 \times 33.0\sigma$
  Initial structure ··· Generation by the density biased Monte Carlo method

- Calculation condition
  Solver ··· Langevin dynamics
  Boundary conditions ··· 2D periodic boundary conditions to the x and y directions. Staggered reflective boundary condition is applied in the plane of $z = Z_{max}$.
  External potential ··· LJ type flat wall potential [**?**] in the plane of $z = 0$.

- Sample files
  Input UDF file ··· "depletion/depletion_in.udf"
  Output UDF file ··· "depletion/out/depletion_out.udf"
  Output data file ··· "depletion/out/ depletion_out.dat"
  Standard output ··· "depletion/out/depletion.log"
  SUSHI output UDF file · "depletion/depletion_sushi_out.udf"

- Notes
  Refer to the sections 2.7.1 and 2.8.

4.1.19 Semi-crystalline lamella

This is an example of modeling semi-crystalline lamellae.

- Calculation model

  1. Bead-spring model
     Molecular model ··· Kremer-Grest type [**?**] bead-spring model
     System size ··· 4 molecules of chain of length, $N = 1200$
     Unit cell ··· $15.89 \times 15.89 \times 20\sigma$
  2. United atom model
     Molecular model ··· United atom model for linear chain alkane[**?**]
     System size ··· 4 molecules of chain of length, $N = 1200$
     Unit cell ··· $10.89 \times 10.89 \times 15\sigma$

Initial structure ··· Generation by the lamella generator

- Calculation conditions
  Solver ··· NVE
  Boundary conditions ··· 3D Periodic boundary conditions

- Sample files

  1. Bead-spring model
     Input UDF file ··· "lamella/bs_lamella_in.udf"
     Molecular structure data UDF file ··· "lamella/bs_lamella_in_str.udf"
     Output UDF file ··· "lamella/out/bs_lamella_out.udf"
     Output data file ··· "lamella/out/bs_lamella_out.dat"
     Standard output ··· "lamella/out/bs_lamella.log"
  2. United atom model
     Input UDF file ··· "lamella/ua_lamella_in.udf"
     Molecular structure data UDF file ··· "lamella/ua_lamella_in_str.udf"
     Output UDF file ··· "lamella/out/ua_lamella_out.udf"
     Output data file ··· "lamella/out/ua_lamella_out.dat"
     Standard output ··· "lamella/out/ua_lamella.log"

- Notes
  Refer to the sections 2.7.1 and 4.7.

4.1.20 Reaction

This is an example of the simulation of creation and scission of chemical bonds.

- Calculation model
  Molecular model ··· Harmonic bond + LJ non-bonding interaction type bead-spring model
  Initial molecular architecture ··· Blend of monoatomic molecules A and B. Number of molecules at initial condition, $M_A = M_B = 50$
  Initial structure ··· Random structure
  Reaction ··· Creation and scission of bond between A-B,A-A,B-B from the blend of monoatomic molecules

- Calculation conditions
  Solver ··· Langevin dynamics
  Boundary condition ··· 3D periodic boundary conditions

- Sample files
  Input UDF file ··· "reaction/react_in.udf"
  Molecular structure data UDF file ··· "reaction/react_in_str.udf"
  Output UDF file ··· "reaction/out/react_out.udf"
  Output data file ··· "reaction/out/react_out.dat"
  Standard output ··· "reaction/out/react.log"

- Notes
  Refer to the sections 2.9 and 4.6.

4.1.21 Polymerization

This is an example of the simulation of polymerization reaction.

- Calculation model
  Molecular model ··· Harmonic bond + LJ non-bonding interaction type bead-spring model
  Initial molecular architecture ··· Blend of momomer M and initiater I. Number of molecules at initial

condition, $M_M = 1000, M_I = 5$
Initial structure $\cdots$ Random structure
Reaction $\cdots$ Sequential polymerization from the initiator

- Calculation conditions
  Solver $\cdots$ Langevin dynamics
  Boundary condition $\cdots$ 3D periodic boundary conditions

- Sample files
  Input UDF file $\cdots$ "polymerization/polymerization_in.udf"
  Molecular structure data UDF file $\cdots$ "polymerization/polymerization_in_str.udf"
  Output UDF file $\cdots$ "polymerization/out/polymerization_out.udf"
  Output data file $\cdots$ "polymerization/out/polymerization_out.dat"
  Standard output $\cdots$ "polymerization/out/polymerization.log"

- Notes
  Refer to the sections 2.9.

4.1.22 DPD

- Calculation model
  Molecular model $\cdots$ A5B5 diblock copolymer
  System size $\cdots$ A5B5 diblock copolymer. Number of molecules, 2,400
  Unit cell $\cdots$ $20.0 \times 20.0 \times 20.0$
  Initial structure$\cdots$ Random

- Calculation conditions
  Solver $\cdots$ DPD
  Boundary conditions $\cdots$ 3D periodic boundary conditions

- Sample files
  Input UDF file $\cdots$ "DPD/A5B5_in.udf"
  Output UDF file $\cdots$ "DPD/out/A5B5_out.udf"
  Output data file $\cdots$ "DPD/out/ A5B5_out.dat"
  Standard output $\cdots$ "DPD/out/A5B5.log"

- Notes
  Refer to the section 2.4

4.1.23 External flow

[Keyword] Velocity field / SLLOD

The external flow obtained by Muffin_phaseseparation is applied to atoms using SLLOD algorithm. This sample reads shear flow caused by the movement of walls. The similar example will be found in the section 5.1.8 of Muffin user's manual.

- Calculation model
  Molecular model $\cdots$ Kremer Grest type [**?**] bead-spring model
  System size $\cdots$ Chain length,$N = 10$. Number of molecules, $M = 650$
  Unit cell $\cdots$ $16.0 \times 16.0 \times 32.0$
  Initial structure $\cdots$ Random

- Calculation conditions
  Solver $\cdots$ NVE
  Boundary conditions $\cdots$ xy-2D periodic boundary conditions
  External field

  - LJ wall potential on xy-plane
  - Velocity field

- Sample files
  Input UDF file ··· “externalflow/externalflow_in.udf”
  Muffin output UDF file ··· “Muffin_shear_out.udf”
  Output UDF file ··· “externalflow/out/externalflow_out.udf”
  Output data file ··· “externalflow/out/ externalflow_out.dat”
  Standard output ··· “externalflow/out/externalflow.log”

- Notes
  Refer to the section 2.6.5

4.1.24 Stress autocorrelation

[Keyword] On the fly

This sample calculate on the fly stress autocorrelation.

- Molecualr model ··· Kremer Grest type [**?**] bead-spring model
  System size ··· 100 chains with the length $N = 40$ Initial structure ··· Random

- Calculation conditions
  Solver ··· NVT_Kremer_Grest
  Boundary conditions ··· 3D periodic boundary conditions

- Sample UDF
  Input UDF ··· “correlation/n40_100_in.udf”
  Restart UDF ··· “correlation/n40_100_rst.udf”
  Output UDF ··· “correlation/out/n40_100_out.udf”
  Output data file ··· “correlation/out/n40_100_out.dat
  Standard output ··· “correlation/out/n40_100.log”

- Notes
  Refer to the section 2.11

4.1.25 MDSCF

[Keyword] MDSCF

Test example of microphase separation of block copolymer using MDSCF proposed by Milano and Kawakatsu [**?**]. Notice that the detail conditions such as temperature control are different from original mothods.

- Molecular model ··· Kremer Grest type [**?**] bead-spring model
  System size ··· 320 chains of A15B15 diblock copolymer chains
  Unit cell ··· $22.436 \times 22.436 \times 22.436$
  Initial structure ··· Random

- Caculation conditions
  Solver ··· NVT_Kremer_Grest
  Boundary conditions ··· 3D periodic boundary conditions
  External field

  - Density_Filed ··· Potential acting on intearction site A based on the density of site B
  - Density_Filed ··· Potential acting on intearction site A based on the density of site B
  - Total_Density_Constrain

  (Note) When the UDF name is not specified in Density Field, field data created at Simulation_Conditions.Density_C
  is used.

- Sample UDF
  Input UDF ··· "MDSCF/a15b15_mdscf_in.udf"
  Output UDF ··· "MDSCF/out/a15b15_mdscf_out.udf"
  Output data file ··· "MDSCF/out/a15b15_mdscf_out.dat"
  Standard output ··· "MDSCF/out/a15b15_mdscf.log"

4.2 (Example I) Simulation of n-alkane using the united atom model

This section shows an example of the simulation of *n*-alkane liquid using the united atom model. The files to be used are placed in "sample/pentaneNVT", if the directory path is not specified.

4.2.1 Preparation of the input data by SILK

SILK is an auxiliary tool which creates the input file of **COGNAC**. Refer to the chapter 6 for the details. In order to use **SILK**, the input UDF file for **SILK** is required, in which the conditions and potential of a simulation are described. The input UDF data for **SILK** has the almost same structure as **COGNAC** input UDF data. The "potential_map.udf" currently prepared as a template is explained, for example.

1. Selection of the input UDF file for **SILK** ("potential_map.udf")

   First, refer to the UDF file for the input of **SILK** ("python/silk/sample/potential_map.udf"). The UDF file "potential_map.udf" consists of a set of records (unit of UDF data). A set of sample of simulation conditions and potential parameters is stored in each record. Here, the first record (the record number is **0** and the record label is "UA_n_ALKANE") is used.

2. Load of the UDF file ("potential_map.udf")

   Start **GOURMET** and open UDF file "potential_map.udf". Display the data and confirm that the record label "UA_n_ALKANE" is displayed on a lower right window, while the data of the record number **0** is displayed on the window.

3. Definition of a set of molecules (Loading, editing and execution of Python **SILK** script)

   Here, a procedure is explained to model the system consisting of 50 molecules of pentane. This system also can be created by the command **SILK_CREATE_LinearPolymer_Semiatomic** of **Action SILK**. See chaper 6 fot the detail.

   (a) Load of Python **SILK** script
   Click a button **Python:Load...** in the lower part of **GOURMET** window and load "python/silk/sample/silk_use_pentane.py" from the appeared file browser.

   (b) Edit of the Python **SILK** script
   Users should edit the section (**SECTION "USER DEFINITION"**) in the loaded script.

   - Output location
     The contents of **SUBSECTION "outputpath"** in **"USER DEFINITION"** are explained.

     * If # is placed at the beginning of a line in Python, the line becomes a comment.

```
####### SUBSECTION "outputpath" ######
def setOutParam(self):
#output Directory (ex. outDir="c:/OCTA8.3/***" (dos), outDir="/home/yourdir/***")
self.engine.outDir="C:/OCTA8.3/ENGINES/COGNAC/python/silk/sample"
#filename without suffix(.udf)
self.engine.cognacFileName="pentane50_in"
#project name
self.engine.prjName="DEVELOP"
#output file is diveded to Structure_data and other Parameters when "TWO_FILES" is cho
```

```
self.engine.fileNumCom="ONE_FILE"
#self.engine.fileNumCom="TWO_FILES"
```

The directory name, where the output file is created, is specified in **self.engine.outDir**.

The output file name is specified in **self.engine.cognacFileName** without the file extension ".udf". In this example, "pentane50_in.udf" is created as **COGNAC** input UDF.

The project name of output file is specified in **self.engine.prjName**.

The flag of whether to create one file or two files for the output of **SILK** (input of **COGNAC**) is specified in **self.engine.fileNumCom**. When **self.engine.fileNumCom** is specified as **ONE_FILE**, the input data for **COGNAC** is output into a single file .
When **self.engine.fileNumCom** is specified as **TWO_FILE**, the input data for **COGNAC** is output into two files. One file has a topological data and the other has the rest of input data. Here, **ONE_FILE** is specified.

- Modeling of molecules
  The contents of **SUBSECTION "system"** in **"USER DEFINITION"** are explained.
  The part of the **SILK** script is as follows.

```
name="alkane"
numAtom=5
numMol=50
atomName= "C"
atomTypeName="atom1"
bondTypeName="bond1"
angleTypeName="angle1"
torsionTypeName="torsion1"
interactionSiteTypeName="siteType1"
self.engine.makeLinPolym(name, numAtom, numMol,
                         atomName, atomTypeName, bondTypeName,
                         angleTypeName, torsionTypeName,
                         interactionSiteTypeName)
```

  The function **makeLinPolym(...)** in the lowest line is a function which creates linear chain molecules.

  The **self.engine.** is an indispensable description in order to use this function.

  The argument of the function **makeLinPolym(...)** is explained below.

  The **name** is a name of molecules.

  The **numAtom** is a number of atoms in a molecule.

  The **numMol** is a number of molecules.

  The **atomName** is a name of atoms in a molecule.

  The **atomTypeName** is a name of atom type, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Atom_Type[]**.
  In this case, **Molecular_Attributes.Atom_Type[0]** is referred by the attribute of atom **C**.

  Figure 4.1 shows the relation between the input UDF and the **SILK** commands.

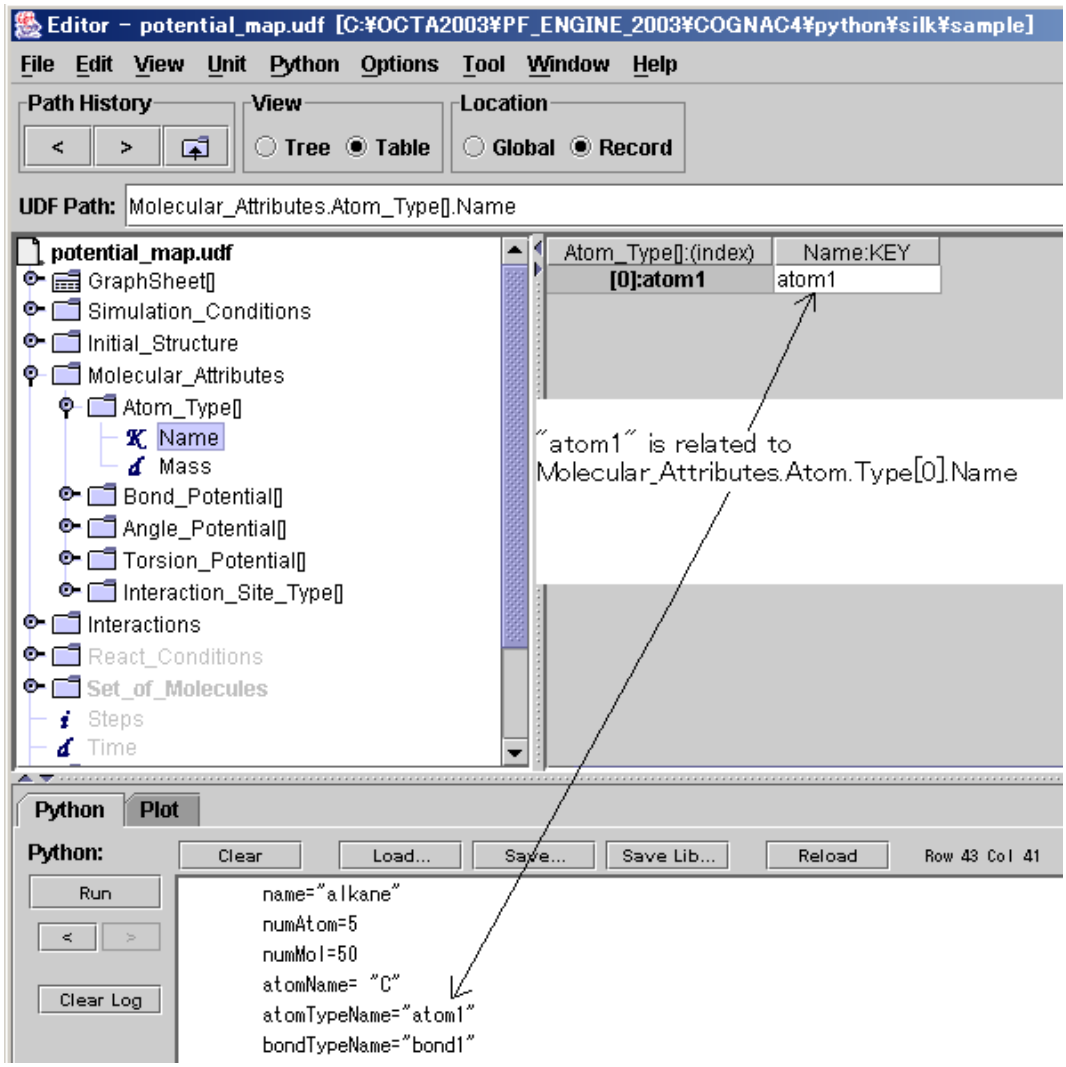

Figure 4.1: Relation of the type attribute of atom

The **bondTypeName** is a name of bond potential, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Bond_Potential**[].
In this case, **Molecular_Attributes.Bond_Potential[0]** is referred by the attribute of bond **C_C**.

Figure 4.2 shows the relation between the input UDF and the **SILK** commands.

The **angleTypeName** is a name of angle potential, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Angle_Potential**[].
In this case, **Molecular_Attributes.Angle_Potential[0]** is referred by the attribute of angle **C_C_C**.

The **torsionTypeName** is a name of torsion potential, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Torsion_Potential**[]. In this case, **Molecular_Attributes.Torsion_Potential[0]** is referred by the attribute of torsion **C_C_C_C**.

The **interactionSiteTypeName** is a name of interaction site type, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Interaction_Site_Type**[]. In this case, **Molecular_Attributes.Interaction_Site_Type[0]** is referred by the attribute of interac-

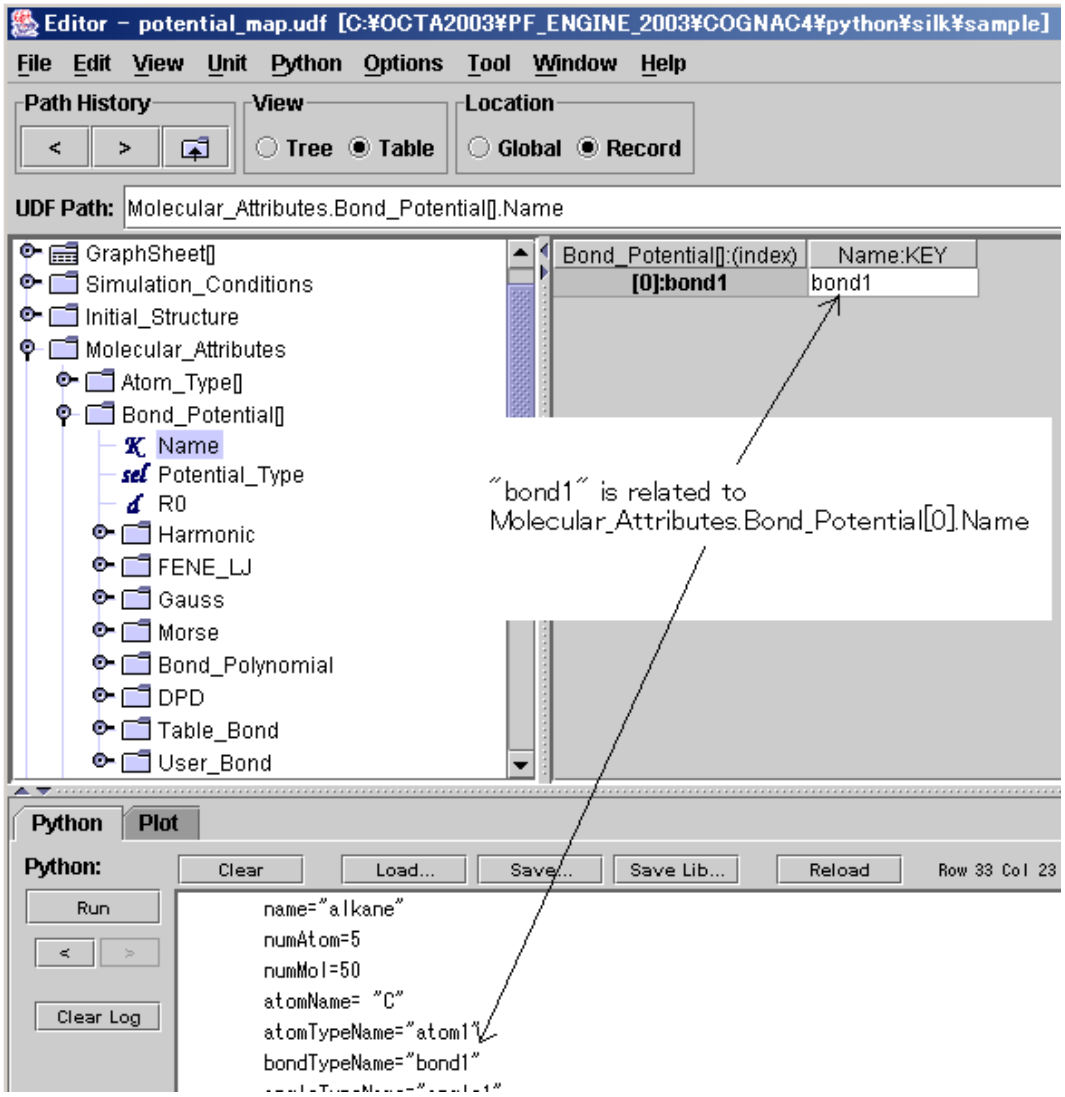

Figure 4.2: Relation of the type attribute of bond

tion site which is defined by a atom **C**.

Figure 4.3 shows the relation between the input UDF and the **SILK** commands.

(c) Execution of Python **SILK** script

After the contents (output place etc.) of the edited Python script are confirmed, execute the script. A result of execution will be displayed in a log window.

4.2.2 Edit of input data, and execution of COGNAC

Load the UDF file created by **SILK** into **GOURMET**, then after confirming the content, execute **COGNAC**.

1. Load of the **COGNAC** input UDF created by **SILK**

   From **GOURMET Editor** window, select **File** → **Open...** and choose "pentane50_in.udf", which is created in the previous section, from the appeared file browser.

2. Confirmation of potential

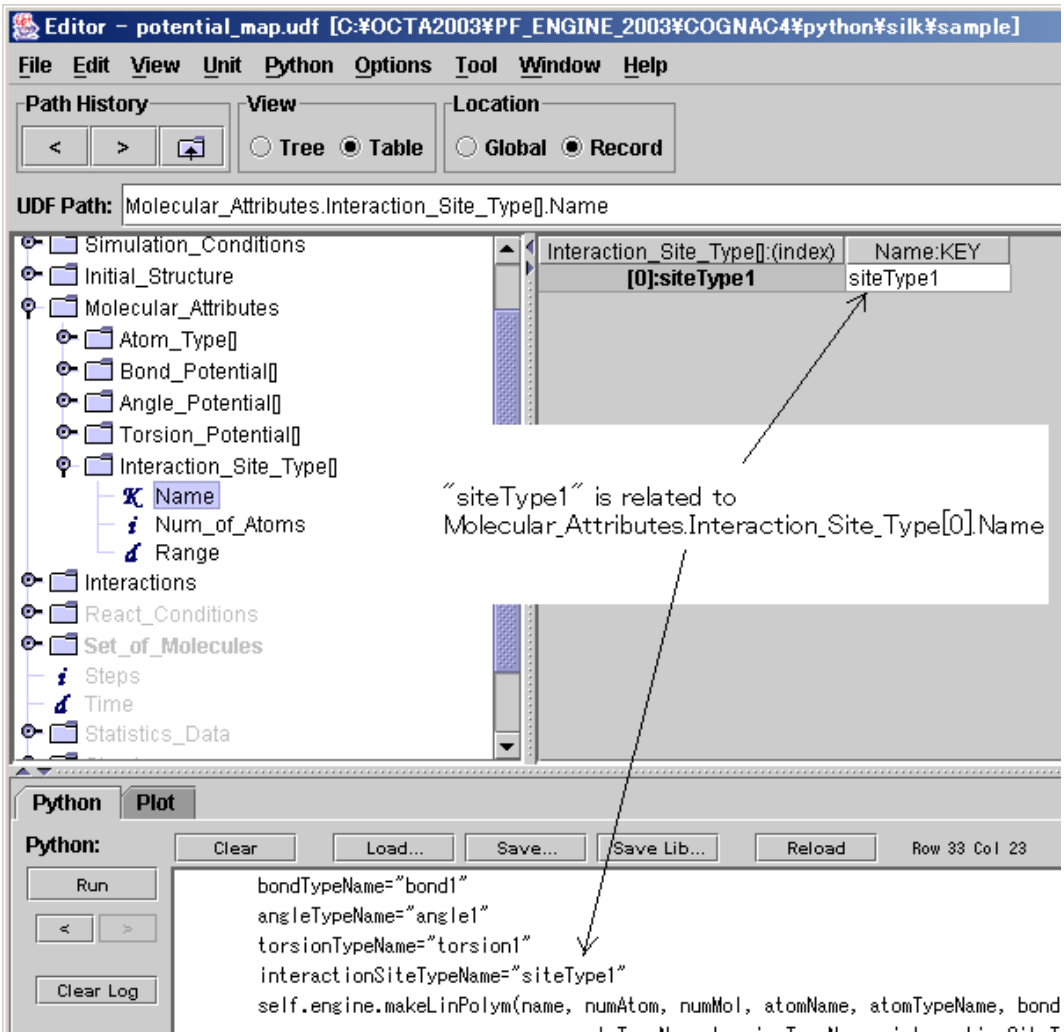

Figure 4.3: Relation of the type attribute of interaction site and pair interaction

The left-hand side of the window, tree-like data structure is displayed. The data of the present record can be shown or edited by clicking this data structure. The parameters related to potentials are described by the following UDF path:

Molecular_Attributes ··· Bonding potential

Interactions ··· Non-bonding and external potential

Unit_Parameter ··· Reduced values of mass, energy and length required in order to perform unit conversion

The data of **Unit_Parameter** is described below.

Mass ··· Unit of mass

The mass ($14amu$) of 1 methylene is defined as a unit mass.

Energy ··· Unit of energy

The parameter $\epsilon(0.40122kJ/mol)$ of the Lennard-Jones potential is defined as a unit energy.

Length ··· Unit of length

The parameter $\sigma(0.40328nm)$ of the Lennard-Jones potential is defined as a unit length.

By defining these units, other units of physical values such as temperature and pressure, are converted. The Python script "python/silk/silk_mujigenkun.py" also can be used for the unit conversion.

3. Setting of the initial structure

   The information about initial structure is set up in the UDF path **Initial_Structure**. The main parameters of **Initial_Structure** are explained:

   Initial_Unit_Cell.Density $\cdots$ Density of the system specified as an initial condition

   The reduced unit value corresponding to $0.9g/cm^3$ is set in this example.

   Generate_Method.Method $\cdots$ Method for initial structure generation

   Random is specified in this example and the data below the UDF Path **Generate_Method.Random** is used.
   The other data (for example, UDF Path **Initial_Structure.Generate_Method.Helix** etc.) which exist in the same layer are ignored.

   Relaxation$\cdots$ Set up for the initial structure relaxation

   When the interaction between molecules is taken into consideration after an initial structure is generated by the **Random** method, excessive repulsive force will arise and a simulation will diverge by the overlap of atoms. In order to prevent it, turn on the relaxation function of initial structure with setting **Relaxation** as **1**.

4. Setting of the calculation conditions

   The calculation conditions are set up by editing the UDF Path **Simulation_Conditions**. The setup parameters are as follows.

   (a) Dynamics condition
   The data in the UDF path **Simulation_Conditions.Dynamics_Conditions** is explained.

   Max_Force$\cdots$ Maximum force allowed to act on each atom during the simulation
   Here, it is set as **10000**.
   Time.delta_T (setting value : **0.001**) $\cdots$ Time of each dynamics step
   Time.Total_Steps (setting value : **2000**) $\cdots$ Total time steps of the simulation
   Time.Output_Interval_Steps (setting value : **20**) $\cdots$ Interval of output
   Temperature.Temperature (setting value : **3.0**) $\cdots$ Target temperature
   Temperature.Interval_of_Scale_Temp (setting value : **10 millions**) $\cdots$ Number of interval steps of velocity scaling

   * Velocity scaling is not performed when **Temperature.Interval_of_Scale_Temp** is larger than set **Time.Total_Steps**. Here, since temperature control is carried out by other technique, there is no necessity of the velocity scaling.

   (b) Solver
   The data in the UDF path **Simulation_Conditions.Solver** is explained.

   Solver_Type (setting value : **Dynamics**) $\cdots$ Selection of dynamics (**Dynamics**) or minimization (**Minimize**)
   Here, **Dynamics** is selected.

   Dynamics.Dynamics_Algorithm (setting value : **NVT_Nose_Hoover**) $\cdots$ Algorithm of dynamics

Here, the Nose_Hoover algorithm for NVT ensemble is selected.

Dynamics.NVT_Nose_Hoover.Q (setting value : **20**) ··· Parameter required for **NVT_Nose_Hoover** algorithm

(c) Boundary conditions

The data in the UDF path **Simulation_Conditions.Boundary_Conditions** is explained.

Boundary_Conditions (setting value : **PERIODIC** in all axes) ··· Boundary condition in each axis of a unit cell

3D periodic boundary conditions are set by setting all of **a_axis**, **b_axis**, and **c_axis** to **PERIODIC**.

(d) Flags for calculating potential terms

The data in the UDF path **Simulation_Conditions.Calc_Potential_Flags** is explained.

Bond (setting value : **1**)
Angle (setting value : **1**)
Torsion (setting value : **1**)
Non_Bonding (setting value : **1**) ··· Setting of the calculation conditions of the bonding, non-bonding and external interactions

When **1** is set in each item, the flag is turned on, and when **0** is set, it is turned off. Here, bond stretching (**Bond**), angle bending(**Angle**), torsion angle (**Torsion**) and non-bonding interaction (**Non_Bonding**) are turned on.

(e) Setting of the output data item

The data in the UDF path **Simulation_Conditions.Output_Flags** is explained.

Statistics ··· Selection of output items, such as temperature and pressure (Selected data are output to data and log file. However, all data are output to UDF file.)

Here, all items are set as **1**.

Structure.Position (setting value : **1**)
Structure.Velocity (setting value : **0**)
Structure.Force (setting value : **0**) ··· Selection of the output about the position, velocity, and force of each atom

Here, only the position of atoms is output.

5. Save of the UDF

After each parameter is edited, the UDF file should be updated from the menu, **File** → **Save** in **GOURMET**.

6. Execution of **COGNAC**

Start from the **GOURMET Engine Run** command with the procedure shown in the chapter 3, or using a command prompt/shell window, move to the directory where UDF files are placed, and execute the following command:

```
cognac92 -I pentane50_in.udf -O pentane50_out.udf > pentane50.log
```

Refer to the section 5.1 about the details of input and output files and the arguments.

After the simulation is completed, "pentane50_out.udf" (UDF output), "pentane50_out.dat" (data output), and "pentane50.log" (log file) are created.

4.2.3 Display and analysis of calculation results

Display of molecular structure

As it is shown in the chapter 3, molecular structure can be displayed by **Action command**. Figure 4.4 shows an example of display with setting **type** to **ball-stick**.

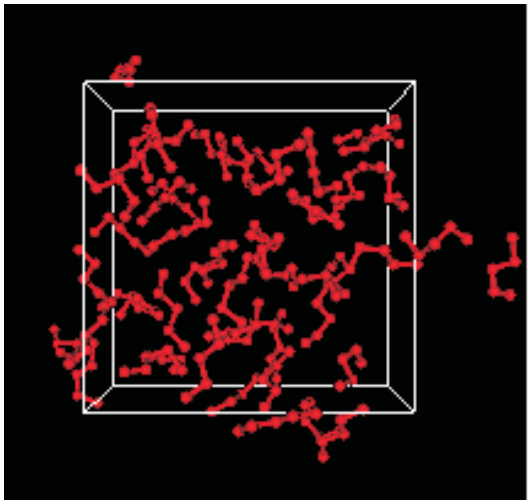

Figure 4.4: Display of molecular structure with **type='ball-stick'**

Display of output data of calculations

Load calculation results into **GOURMET** and display data.

1. Load of the output UDF

   In **GOURMET Editor** window, select **File** → **Open...** and choose "pentane50_out.udf" from the file browser. It is unnecessary if the UDF file has already been loaded.

2. Display of input data

   The input parameters used for the calculation are output in the output UDF of **COGNAC**. Therefore, the input data can be displayed in addition to the output data.

   (Example)
   If **Simulation_Condition** → **Dynamics_Condition** → **Time** is selected, the input value of Δt, total step, and the interval of output of simulation will be displayed.

3. Display of output data

   In order to display output data, specify a record number by moving a slide bar or setting an integer in a lower right of the window.

   (Example)
   The section averages of the energy of each functional term are displayed by selecting **Statistics_Data**

→ **Energy** → **Batch_Average**.

Plotting of the results

Make a plot of temperature and pressure as a function of time from output data.

1. Load of the output UDF

   Read output UDF as described above. It is unnecessary if the UDF file has already been loaded.

2. Collection of the data of every record as a function of time

   Select **Python:Load** first, and load "MakeGraphSheet.py" from the appeared file browser and execute the script. Then selecting **GraphSheet** which is shown on the top of the tree in the left part of window at **Table mode**, the data of **Time**, **Temperature** and **Pressure** for every record are shown in one sheet.

3. Plotting of the temperature and pressure as a function of time

   Select **Plot** from the tag **Python/Plot** which is in the lower left of the window after **GraphSheet** is selected in the UDF tree of the screen. Then select **Plot:Make** in the **Plot** menu. The command line of gnuplot appears in a window. If the plot command is executed using the command lines, the index of **GraphSheet** will become a horizontal axis. Therefore, rewrite the command line as follows.

```
(original)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 1:2 title 'Time' with lines , \
     "plot.dat" using 1:3 title 'Temperature' with lines , \
     "plot.dat" using 1:4 title 'Pressure' with lines

(after modification: when plotting temperature)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 2:3 title 'Temperature' with lines

(after modification: when plotting pressure)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 2:4 title 'Pressure' with lines
```

4. Execution of gnuplot

   The gnuplot is executed by selecting **Plot:Plot**. And as it is shown in Figure4.5, temperature or pressure as a function of time is plotted.

4.3 (Example II) Simulation of poly (ethylene oxide) (PEO)

This section shows an example of the MD simulation of poly(ethylene oxide) (PEO). Especially, the construction method of polymer chains with complicated sequence such as PEO by **SILK** is explained.

The files to be used are placed in "sample/peo", if the directory path is not specified.

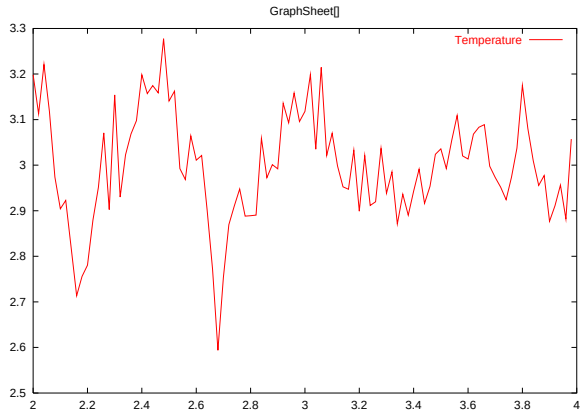

Figure 4.5: Plot of temperature as a function of time

4.3.1 Preparation of the input data by SILK

SILK is an auxiliary tool which creates the input file of **COGNAC**. Refer to the chapter 6 for the details. In order to use **SILK**, the input UDF file for **SILK** is required, in which the conditions and potential of a simulation are described. The input UDF data for **SILK** has the almost same structure as **COGNAC** input UDF data. The "potential_map.udf" currently prepared as a template is explained, for example.

1. Selection of the input UDF file for **SILK** ("potential_map.udf")

   First, refer to the UDF file for the input of **SILK** ("python/silk/sample/potential_map.udf"). The UDF file "potential_map.udf" consists of a set of records. A set of sample of simulation conditions and potential parameters is stored in each record. Here, the 3rd record (the record number is **2** and the record label is "UA_PEO_LiI") is used.

2. Load of the UDF file ("potential_map.udf")

   Start **GOURMET** and open UDF "potential_map.udf". Since the data of the record number **0** is displayed on the window, move the slide bar under the **GOURMET** window right until the record label "UA_PEO_LiI" is displayed on a lower right window.

3. Definition of a set of molecules (Loading, editing and execution of Python **SILK** script)

   Here, a procedure is explained to model the system consisting of PEO chains and LiI molecules.

   (a) Load of Python **SILK** script
   Click a button **Python:Load...** in the lower part of **GOURMET** window and load "python/silk/sample/silk_use_peo.py" from the appeared file browser.

   (b) Edit of the Python **SILK** script
   Users should edit the section (**SECTION "USER DEFINITION"**) in the loaded script.

   - Output location
     The contents of **SUBSECTION "outputpath"** in **"USER DEFINITION"** are explained.

     * If # is placed at the beginning of a line in Python, the line becomes a comment.

```
####### SUBSECTION "outputpath" ######
def setOutParam(self):
#output Directory (ex. outDir="c:/OCTA8.3/***" (dos), outDir="/home/yourdir/***")
self.engine.outDir="C:/OCTA8.3/ENGINES/COGNAC/python/silk/sample"
#filename without suffix(.udf)
self.engine.cognacFileName="peo_in"
#project name
self.engine.prjName="DEVELOP"
#output file is diveded to Structure_data and other Parameters when "TWO_FILES" is cho
#self.engine.fileNumCom="ONE_FILE"
self.engine.fileNumCom="TWO_FILES"
```

The directory name, where the output file is created, is specified in **self.engine.outDir**.

The output file name is specified in **self.engine.cognacFileName** without the file extension ".udf". In this example, "peo_in.udf" is created as **COGNAC** input UDF.

The project name of output file is specified in **self.engine.prjName**.

The flag of whether to create one file or two files for the output of **SILK** (input of **COGNAC**) is specified in **self.engine.fileNumCom**. When **self.engine.fileNumCom** is specified as **ONE_FILE**, the input data for **COGNAC** is output into a single file .
When **self.engine.fileNumCom** is specified as **TWO_FILE**, the input data for **COGNAC** is output into two files. One file has a topological data and the other has the rest of input data.

- Modeling of molecules (1)

  SUBSECTION "system" in "USER DEFINITION" is explained.
  The part of the **SILK** script, which defines PEO molecules, is as follows.

```
##### Build system(Make sure to ...
numAtom=36
numMol=5
self.engine.createMolecule("peo")
for i in range(0, numAtom):
 if(i%3==1):
  self.engine.addAtoms("peo", "O", "atom2")
 else:
  self.engine.addAtoms("peo", "C", "atom1")
for i in range(0, numAtom-1):
 if(i%3==2):
  self.engine.addBonds("peo", i, i+1, "bond2")
 else:
  self.engine.addBonds("peo", i, i+1, "bond1")
for i in range(0, numAtom-2):
 if(i%3==0):
  self.engine.addAngles("peo", i, i+1, i+2, "angle1")
 else:
  self.engine.addAngles("peo", i, i+1, i+2, "angle2")
for i in range(0, numAtom-3):
 if(i%3==1):
  self.engine.addTorsions("peo", i, i+1, i+2, i+3, "torsion2")
 else:
  self.engine.addTorsions("peo", i, i+1, i+2, i+3, "torsion1")
```

```
for i in range(0, numAtom):
 if(i%3==1):
  self.engine.addInteractionSites("peo", [i], "siteType2", "PAIR")
 else:
  self.engine.addInteractionSites("peo", [i], "siteType1", "PAIR")
for i in range(0, numAtom):
 if(i%3==1):
  self.engine.addInteractionSites("peo", [i], "POINT_CHARGE", "COULOMB", -9.1089)
 else:
  self.engine.addInteractionSites("peo", [i], "POINT_CHARGE", "COULOMB", 4.5545)
self.engine.setSystem("peo", numMol)
```

In order to define the topology of PEO molecules, some basic functions of **SILK** are used. Although the detail of Python syntax is not given, each function and argument is explained below.

i. Registration of the name of molecules and the number of atoms in a molecule:

```
numAtom=36
```

The **numAtom** is a number of atoms in a molecule and it is set as **36**. In this example, $CH_2 - O - CH_2$ fragment is defined as one unit in PEO molecules, and one chain consists of 12 units. Since CH_2 is modeled as 1 atom, the chain consists of 36 atoms.

```
self.engine.createMolecule("peo")
```

Here, the function **createMolecule("peo")** is used to register a molecule. This function registers a molecule to **SILK** with the name **peo**.

ii. Registration of atoms to the registered molecule (name of the molecule : **peo**)

```
for i in range(0, numAtom):
 if(i%3==1):
  self.engine.addAtoms("peo", "O", "atom2")
 else:
  self.engine.addAtoms("peo", "C", "atom1")
```

The **if** statement is executed in the **for** loop from **i=0** to **i=35**(**numAtom**). The content of the **if** statement is as follows.
When the remainder of **i** by **3** is **1**, the function **addAtoms("peo", "O", "atom2")** of **SILK** is called, and the Oxygen atom in $CH_2 - O - CH_2$ fragment is registered to the molecule **peo**.
When the remainder is not **1**, the function **addAtoms("peo", "C", "atom1")** of **SILK** is called, and the methylene unit CH_2 (methyl unit CH_3) is registered to the molecule **peo**.

iii. Registration of bonds to the molecule (name of the molecule : **peo**)

```
for i in range(0, numAtom-1):
 if(i%3==2):
  self.engine.addBonds("peo", i, i+1, "bond2", which is created in the previous
  section,)
 else:
  self.engine.addBonds("peo", i, i+1, "bond1")
```

The **if** statement is executed in the **for** loop from **i=0** to **i=34** (**numAtom**-1). The content of the **if** statement is as follows.
When the remainder of **i** by **3** is **2**, the function **addBonds("peo", i, i+1, "bond2")** of **SILK** is called, and the bond between CH_2-CH_2 in CH_3-O-CH_2-CH_2-O-CH_2 is registered to the molecule **peo**.
The arguments **bond1** and **bond2** used in the functions correspond to the name of bond potential type. For example, **Bond_Potential** with the name of **bond1** (in this case, the UDF path "Molecular_Attributes.Bond_Potential[0]") is referred by the attribute of bond **bond1**.

iv. Registration of angles and torsions to the molecule (name of the molecule : **peo**)

```
for i in range(0, numAtom-2):
 if(i%3==0):
  self.engine.addAngles("peo", i, i+1, i+2, "angle1")
 else:
  self.engine.addAngles("peo", i, i+1, i+2, "angle2")
for i in range(0, numAtom-3):
 if(i%3==1):
  self.engine.addTorsions("peo", i, i+1, i+2, i+3, "torsion2")
 else:
  self.engine.addTorsions("peo", i, i+1, i+2, i+3, "torsion1")
```

Angles and torsions can be registered by almost the same way as the registration of bonds. The functions **addAngles (...)** and **addTorsions (...)** are prepared to register angles and torsions respectively.

v. Registration of interaction sites to the molecule (name of the molecule : **peo**)

```
for i in range(0, numAtom):
 if(i%3==1):
  self.engine.addInteractionSites("peo", [i], "siteType2", "PAIR")
 else:
  self.engine.addInteractionSites("peo", [i], "siteType1", "PAIR")
```

The interaction sites are registered for non-bonding interaction. The function **addInteractionSites(...)** is used to register the interaction sites in **SILK**.
An argument [**i**] is an array of atom. Since the interaction sites can be defined by two or more atoms in **COGNAC**, the atom(s) are specified by an array. And, the 4th argument, **"PAIR"** specifies that the interaction site is registered for pair interaction.

vi. Registration of electrostatic sites to the molecule (name of the molecule : **peo**)

```
for i in range(0, numAtom):
 if(i%3==1):
  self.engine.addInteractionSites("peo",[i],"POINT_CHARGE","COULOMB",-9.1089)
 else:
  self.engine.addInteractionSites("peo",[i],"POINT_CHARGE","COULOMB",4.5545)
```

The electrostatic sites are registered. The function **addInteractionSites(...)** is used to register the electrostatic sites in **SILK**.
The 4th argument **"COULOMB"** specifies that the interaction site is defined for electrostatic interaction. And, the data in the UDF path **Interactions.Electrostatic_Interaction**[**0**], which have the name **"POINT_CHARGE"**, is used for the calculation.
A charge value or dipole moment is set in the 5th argument.

vii. Specification of the molecules for the output from the registered molecules (name of the molecule : **peo**)

```
self.engine.setSystem(name, numMol)
```

Here, the number of molecules to be created from the registered molecule (name : **peo**) is specified. Notice that the data is not output unless the function **setSystem(...)** is called even if the molecular data is registered to **SILK**.

- Modeling of molecules (2)
  Here, the part of the **SILK** script which defines Li ion and I ion is explained. They are modeled as monoatomic molecules.

  i. Registration of 5 Li ions

```
#Li ion
numMol=5
self.engine.createMolecule("Li")
self.engine.addAtoms("Li", "Li", "atom3")
self.engine.addInteractionSites("Li", [0], "siteType3", "PAIR")
self.engine.addInteractionSites("Li", [0], "POINT_CHARGE", "COULOMB", 18.2178)
self.engine.setSystem("Li", numMol)
```

  ii. Registration of 5 I ions

```
#I ion
numMol=5
self.engine.createMolecule("I")
self.engine.addAtoms("I", "I", "atom4")
self.engine.addInteractionSites("I", [0], "siteType4", "PAIR")
self.engine.addInteractionSites("I", [0], "POINT_CHARGE", "COULOMB", -18.2178)
self.engine.setSystem("I", numMol)
```

(c) Execution of the Python **SILK** script
After the contents (output place etc.) of the edited Python script are confirmed, execute the script. A result of execution will be displayed in a log window.

4.3.2 Edit of input data, and execution of COGNAC

Load the UDF file created by **SILK** into **GOURMET**, then after editing the data, execute **COGNAC**.

1. Load of the **COGNAC** input UDF created by **SILK**
   From **GOURMET Editor** window, select **File** → **Open...** and choose "peo_in.udf", which is created in the previous section, from the appeared file browser.

2. Confirmation of potential
   The left-hand side of the window, tree-like data structure is displayed. The data of the present record can be shown or edited by clicking this data structure.
   In order to perform unit conversion, the reduced values of mass, energy and length are set into the UDF path **Unit_Parameter**. This example uses the potential of the united atom model of PEO, which is developed by Jorgensen and others[**?**, **?**]. The data of **Unit_Parameter** is described below.

Mass ··· Unit of mass

$10[amu]$ is defined as a unit mass.

Energy ··· Unit of energy

$1[kcal/mol](4.18605kJ/mol)$ is defined as a unit energy.

Length ··· Unit of length

$1\text{Å}(0.1nm)$ is defined as a unit length.

By defining these units, other units of physical values such as temperature and pressure can be converted on **GOURMET**. The Python script "python/silk/silk_mujigenkun.py" also can be used for the unit conversion. The unit of charge also can be converted by this script.

3. Setting of the initial structure

   The information about initial structure is set up in the UDF path **Initial_Structure**. Here, select **Restart** in **Initial_Structure.Generate_Method.Method**, then the restart UDF name is set as "peo_rst.udf".

4. Setting of the calculation conditions

   The calculation conditions are set up in the UDF path **Simulation_Conditions**. The setup parameters are as follows.

   (a) Dynamics condition

   The data in the UDF path **Simulation_Conditions.Dynamics_Conditions** is explained.

   Time.delta_T (setting value : **0.01617**) ··· Time of each dynamics step (corresponding to 2.5 fs)

   Time.Total_Steps(setting value : **100,000**) ··· Total time steps of the simulation

   Time.Output_Interval_Steps (setting value : **200**) ··· Interval of output

   In order to analyze the diffusion behavior of Li ion, change a simulation time step to perform a simulation for longer time scale. Open **Simulation_Conditions.Dynamics_Conditions.Time** and change **Total_Steps** to **100,000** as shown in Figure 4.6. Set **Output_Interval_Steps** as **200** and increase a number of sampling to obtain precise results.

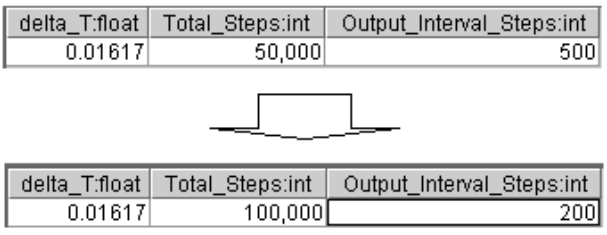

Figure 4.6: Change of **Total_Steps and Output_Interval_Steps**

Temperature.Temperature (setting value : **0.715**) ··· Target temperature (corresponding to 360 K)

Temperature.Interval_of_Scale_Temp (setting value : **0**) ··· Number of interval steps of velocity scaling

* Velocity scaling is not performed when **Temperature.Interval_of_Scale_Temp** is set as **0**. Here, since temperature control is carried out by other technique, there is no necessity of the velocity scaling.

(b) Solver
The data in the UDF path **Simulation_Conditions.Solver** is explained.

Solver_Type (setting value : **Dynamics**) ··· Selection of dynamics (**Dynamics**) or minimization (**Minimize**)
Here, **Dynamics** is selected.

Dynamics.Dynamics_Algorithm (setting value : **NVT_Nose_Hoover**) ··· Algorithm of dynamics
Here, the Nose Hoover algorithm for NVT ensemble is selected.

Dynamics.NVT_Nose_Hoover.Q (setting value : **20**) ··· Parameter required for **NVT_Nose_Hoover** algorithm

(c) Boundary conditions
The data in the UDF path **Simulation_Conditions.Boundary_Conditions** is explained.

Boundary_Conditions (setting value : **PERIODIC** in all axes) ··· Boundary condition in each axis of a unit cell
3D periodic boundary conditions are set by setting all of **a_axis**, **b_axis** and **c_axis** to **PERIODIC**.

(d) Flags for calculating potential term
The data in the UDF path **Simulation_Conditions.Calc_Potential_Flags** is explained.

Bond (setting value : **1**)
Angle (setting value : **1**)
Torsion (setting value : **1**)
Non_Bonding (setting value : **1**)
Electrostatic (setting value : **1**) ··· Setting of the calculation conditions of the bonding, non-bonding and external interaction

When **1** is set in each item, the flag is turned on, and when **0** is set, it is turned off. Here, bond stretching (**Bond**), angle bending(**Angle**), torsion angle (**Torsion**), non-bonding interaction (**Non_Bonding**) and electrostatic interaction (**Electrostatic**) are turned on.

(e) Setting of the output data item
The data in the UDF path **Simulation_Conditions.Output_Flags** is explained.

Statistics ··· Selection of output items, such as temperature and pressure (Selected data are outputted to data and log file. However, all data are outputted to UDF file.)
Here, all items are set as **1**.

Structure.Position (setting value : **1**)
Structure.Velocity (setting value : **0**)
Structure.Force (setting value : **0**) ··· The selection of the output about the position, velocity, and force of each atom

Here, only position of atoms is output.

5. Save of the UDF

   After each parameter is edited, the UDF file should be updated from the menu, **File** → **Save** in **GOURMET**.

6. Execution of **COGNAC**

   Start from the **GOURMET Engine Run** command with the procedure shown in the chapter 3, or using a command prompt/shell window, move to the directory where UDF files are placed, and execute the following command:

```
cognac92 -I peo_in.udf -O peo_out.udf > peo.log
```

   After the simulation is completed, "peo_out.udf" (UDF output), "peo_out.dat" (data output), and "peo.log" (log file) are created.

4.3.3 Display and analysis of calculation results

Display of molecular structure

As it is shown in the chapter 3, molecular structure can be displayed by **Action command**.

Analysis of mean square displacements (MSD)

Calculate the mean square displacement (MSD) of Li ion in order to analyze the diffusion of ions. **Action command : ANALYSIS_msd...** can be used to execute the same analysis. See details in section 7.5.

1. Load of the output UDF

   In **GOURMET Editor** window, select **File** → **Open...** and choose "peo_out.udf" from the appeared file browser.

2. Load and execution of Python script

   Select **Python:Load...** and load "msd.py" from the appeared file browser. Then execute the script by selecting **Run** button. After the script is executed, each component of MSD of Li ion as a function of time will be set in **GraphSheet**, which is shown at **Table mode**.

3. Plotting of MSD

   Select **Plot** from the tag **Python/Plot** which is in the lower left of the window after **GraphSheet** is selected in the UDF tree of the screen. Then select **Plot:Make** in the plot menu. The command line of gnuplot appears in a window. If the plot command is executed using the command line, the index of **GraphSheet** will become a horizontal axis. Therefore, rewrite the command line as follows.

```
(original)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 1:2 title 'Time' with lines , \
     "plot.dat" using 1:3 title 'msd' with lines , \
     "plot.dat" using 1:4 title 'msd_x' with lines , \
```

```
     "plot.dat" using 1:5 title 'msd_y' with lines , \
     "plot.dat" using 1:6 title 'msd_z' with lines

(after modifiaction)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 2:3 title 'msd' with lines , \
     "plot.dat" using 2:4 title 'msd_x' with lines , \
     "plot.dat" using 2:5 title 'msd_y' with lines , \
     "plot.dat" using 2:6 title 'msd_z' with lines
```

When **Plot:Plot** is selected after that, each component of axes(x, y and z) of MSD and MSD are plotted as shown in Figure 4.7.

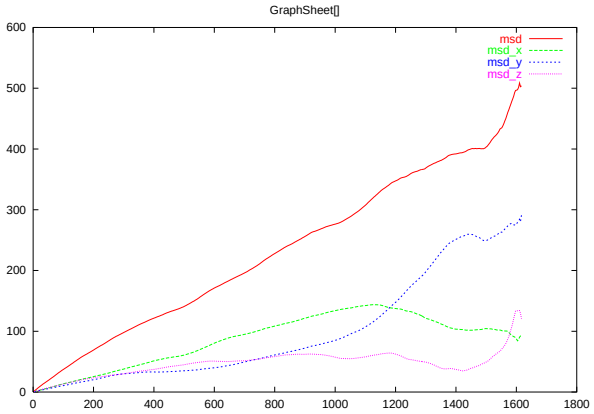

Figure 4.7: Plot of mean square displacement

4.4 (Example III) Simulation of ABA triblock copolymer lamella

This section shows an example of modeling lamella structures of block copolymer from the information obtained by **SUSHI**. Since the molecular model is the same as that used in the chapter 3, refer to the chapter for the preparation of molecular model by **SILK**. The files to be used are placed in "sample/blend", if the directory path is not specified.

The example of zooming between **SUSHI-COGNAC** is described also in the "AMUSE" document, please refer to it.

4.4.1 Calculation conditions of SUSHI

The calculation conditions of **SUSHI** can be checked from the output file of **SUSHI**. Refer to "SUSHI user's manual" for the details. It is necessary to output **segment_volume_fraction** of every segments for the input of **COGNAC**. Other input parameters are the same as those of the usual calculations of block copolymers. Refer to "SUSHI user's manual" about the UDF data of **SUSHI**.

4.4.2 Edit of input data, and execution of COGNAC

Load the UDF file prepared in the chapter 3 into **GOURMET**, then after editing the data, execute **COGNAC**.

Since the input UDF "A20B40A20_zoom_in.udf" is already prepared in a sample directory, refer to it too.

1. Load of the **COGNAC** input UDF

   From **GOURMET Editor** window, select **File** → **Open...** and choose "A20B40A20_in.udf" from the appeared file browser.

2. Setting for the density biased potential

   Set up **External_Interaction**, in order to apply the density biased potential.

   When **Interactions.External_Interaction[]** is opened first, an empty row for array elements will be displayed as in Figure 4.8.

External_Inte...	Name:KEY	Potential_Ty...	Site_Name:...	LJ_Wall:use...	LJ_
[]	-	-	-	{...}	

Figure 4.8: Initial state of **External_Interaction[]**

After the empty row is pointed, perform **Edit** → **Insert Row** twice, and create two rows of **External_Interaction**.

And set parameters into each element, i.e. set the **Name**, **Potential_Type** and **Site_Name**, for external interactions, as shown in Figure 4.9.

External_Inte...	Name:KEY	Potential_Ty...	Site_Name:...	LJ_Wall:use...	LJ_
[0]	Density_AB	Density_Field	siteType1	{...}	
[1]	Density_BA	Density_Field	siteType2	{...}	

Figure 4.9: Input of **External_Interaction**

Next, open **Interactions.External_Interaction[].Density_Field**, and select the type of external potential, **Density_Biased_Potential**. Then set the output UDF file of **SUSHI** to **UDF_Name** and Index of segment type used in the **SUSHI** calculation to **Component_Index** for the external potential, as shown in Figure 4.10.

External_Inte...	UDF_Name:string	Component_...	Potential_Type:select	Densi
[0]:Density...	A20B40A20_sushi_out.udf	1	Density_Biased_Potential	{
[1]:Density...	A20B40A20_sushi_out.udf	0	Density_Biased_Potential	{

Figure 4.10: Input of **Density_Field**

Next, set the intaraction parameter **chi** in **Interactions.External_Interaction[].Density_Field.Density_Biased_Potential**

Notice that **chi** value in the second row is a twice of χ parameter used in **SUSHI**, since **Segment_ID = 0** corresponds to the head **A** block of ABA triblock copolymer and the volume fraction is a half of the total volume fractions of segment type **A**.

3. Change of the shape of the unit cell

   Change the shape of the unit cell of **COGNAC** to correspond to the lamella period and cell size which were calculated by **SUSHI**.

External_Interaction[...	chi:double
[0]:Density_AB	1.0
[1]:Density_BA	2.0

Figure 4.11: Input of **Density_Biased_Potential**

Input **39.0** in **Initial_Structure.Initial_Unit_Cell.Cell_Size.c**, which is the same as the mesh size used by **SUSHI**. When **c** is set as **39.0** and **a** and **b** are set as **0**, cell length **a** and **b** are calculated from the specified density.

4. Setting for the density biased Monte Carlo method

   Set up the input parameters for the node density biased Monte Carlo to generate initial structures.
   Open **Initial_Structure.Generate_Method.Random.Node_Density_Bias**[], then perform **Edit** → **Insert Row** like the case of the setup of **External_Interaction**, and make a row for input.
   Next, set parameters to each item, as shown in Figure 4.12.
   The **Molecular_Name** is a molecular name to apply the density biased Monte Carlo method. The **UDF_Name** is a UDF file name of **SUSHI** output. The elements of segment volume fraction from the **Start_ID** to the **End_ID** in **SUSHI** output are used for generating **A20B40A20** chain.

Node_Densi...	Molecular_N...	UDF_Name:string	Start_ID:int	End_ID:int	Scale_Para...	Max_Retry:int
[0]	A20B40A20	A20B40A20_sushi_out.udf	0	79	1.0	10,000

Figure 4.12: Input of **Node_Density_Bias**

5. Save of the UDF

   After each parameter is edited, the UDF file should be saved as another file name (for example, "A20B40A20_zoom_in.udf") from the menu, **File** → **Save As...** in **GOURMET**.

6. Execution of **COGNAC**

   Start from the **GOURMET Engine Run** command with the procedure shown in the chapter 3, or using a command prompt/shell window, move to the directory where UDF files are placed, and execute the following command:

```
cognac92 -I A20B40A20_zoom_in.udf -O A20B40A20_zoom_out.udf > A20B40A20_zoom.log
```

   After the simulation is completed
   "A20B40A20_zoom_out.udf" (UDF output), "A20B40A20_zoom_out.dat" (data output) and "A20B40A20_zoom.log" (log file) are created.

   When starting **COGNAC** from the **GOURMET Engine Run** command, **SUSHI** UDF file must be specified in **Parms UDF:** of **(Local input Files)**.

4.4.3 Display and analysis of calculation results

Display of molecular structure

As it is shown in the chapter 3, molecular structure can be displayed by **Action command**.

Analysis of pair distribution functions

As shown in the chapter 3, the output UDF is loaded into **GOURMET**, and pair distribution function $g_{AB}(r)$ can be calculated and plotted. Since this example generates lamella structure, the result is different from that of random structure in the chapter 3.

Analysis of density profiles

Calculate and plot 1D profiles of the volume fraction of each segment. **Action command : ANALYSIS_1D_profile...** can be used to execute the same analysis. See details in section 7.5.

1. Load of the output UDF

   From **GOURMET Editor** window, select **File** → **Open...** and choose "A20B40A20_zoom_out.udf" from the appeared file browser. It is unnecessary if the UDF file has already been loaded.

2. Change of the current record

   Move the slide bar under the window to the last record (record number is **11**).

3. Load and execution of Python script

   Select **Python:Load...** and load "density1D.py" from the appeared file browser. After the script is executed, the position Z, the volume fraction of segment type A and B, $\phi_A(Z)$ and $\phi_B(Z)$ will be set in **GraphSheet**, which is shown at **Table mode**.

4. Plotting of $\phi_A(Z)$ and $\phi_B(Z)$

   Select **Plot** from the tag **Python/Plot** which is in the lower left of the window after **GraphSheet** is selected in the UDF tree of the screen. Then select **Plot:Make** in the plot menu. The command line of gnuplot appears in a window. If the plot command is executed using the command line, the index of **GraphSheet** will become a horizontal axis. Therefore, rewrite the command line as follows.

```
(original)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 1:2 title 'Z' with lines , \
     "plot.dat" using 1:3 title 'phi_A' with lines , \
     "plot.dat" using 1:4 title 'phi_B' with lines

(after modification)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 2:3 title 'phi_A' with lines , \
     "plot.dat" using 2:4 title 'phi_B' with lines
```

   When **Plot:Plot** is selected after that, volume fractions $\phi_A(Z)$ and $\phi_B(Z)$ are plotted as shown in Figure 4.13.

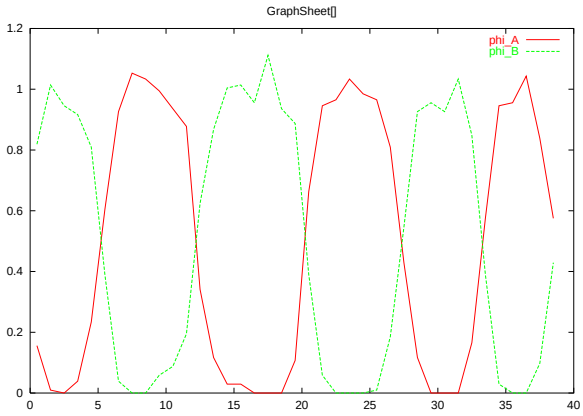

Figure 4.13: Plot of volume fractions

4.5 (Example IV) Simulation using table potential

This section shows an example which uses numerical table data for interaction potential. Since the molecular model and the interactions are the same as those in the section 4.2, refer to the section for the preparation of molecular model by **SILK**.

The files to be used are located in "sample/tablepotential", if the directory path is not specified.

4.5.1 Confirmation of potential tables

Load the UDF file for the bond potential and confirm the contents.

1. Load of the UDF file "bond_table.udf"

   From **GOURMET Editor** window, select **File** → **Open...** and choose "bond_table.udf" from the appeared file browser.

2. Confirmation of minimum, maximum and the number of division

   As it is shown in Figure 4.14, open **Mesh.axes[].values[]** and confirm the following data

   value[0] ··· minimum length
   value[1] ··· maximum length
   value[2] ··· number of divisions

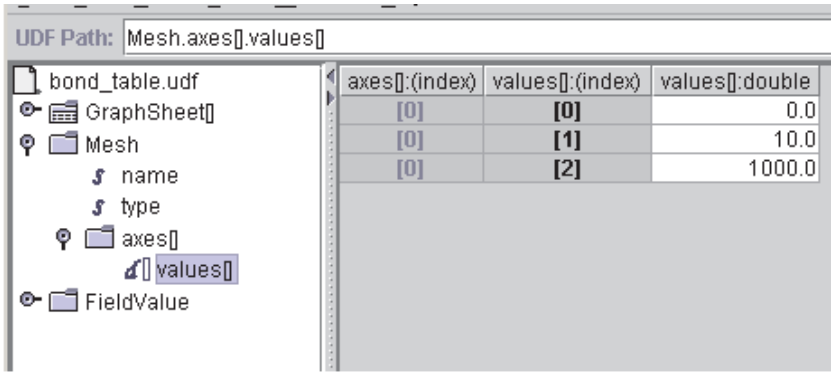

Figure 4.14: **Bond_Potential_Table**

3. Confirmation of the potential

   Open **FieldValue.value**[] and confirm the data. The values of potential energy from the minimum distance to maximum distance are set.

4.5.2 Edit of input data, and execution of COGNAC

Load the UDF file prepared in the section 4.2 into **GOURMET**, then after editing the data, execute **COGNAC**.

Since the input UDF "pentane_table_in.udf" is already prepared in a sample directory, refer to it too.

1. Load of the input UDF

   From **GOURMET Editor** window, select **File** → **Open...** and choose "pentaneNVT_in.udf" from the appeared file browser.

2. Change of potential type, and specification of the UDF file for table potential

   Setting of the bond potential is explained, for example.
   Open **Molecular_Attributes.Bond_Potential[0]** and change the **Potential_Type** to **Table_Bond** as it is shown in Figure 4.15.

Bond_Potent..	Name:KEY	Potential_Type:select	R0:float	Table_Bond:...
[0]	bond1	Table_Bond	0.4	{...}

Figure 4.15: Change of **Bond_Potential**

   Next, open **Molecular_Attributes.Bond_Potential[0].Table_Bond** and specify the UDF file name for potential table to **UDF_Name** and an order of interpolation to **Order** as shown in Figure 4.16.

Bond_Potent..	UDF_Name:string	Order:int
[0]	bond_table.udf	2

Figure 4.16: Setup of **Table_Bond**

   Set up **Angle_Potential**[], **Torsion_Potential**[] and **Interactions.Pair_Interaction**[] in the same way.

3. Save of the UDF

   After each parameter is edited, the UDF file should be saved as another file name (for example, "pentane50_table_in.udf") from the menu, **File** → **Save As...** in **GOURMET**.

4. Execution of **COGNAC**

   In the command prompt/shell window, move to the directory where UDF files are placed, and execute the following command:

```
cognac92 -I pentane50_table_in.udf -O pentane50_table_out.udf > pentane50_table.log
```

   After the simulation is completed,
   "pentane50_table_out.udf" (UDF output), "pentane50_table_out.dat" (data output)
   and "pentane50_table.log" (log file) are created.

4.5.3 Display and analysis of calculation results

Display of molecular structure

As it is shown in the chapter 3, molecular structure can be displayed by **Action command**.

Plotting of the results

As shown in the section 4.2, plot temperature and pressure as a function of time. Compare the results with those of the section 4.2 and confirm that the results are almost the same between them.

4.6 (Example V) Simulation with chemical reaction

This section shows an example of the simulation with creation and scission of chemical bonds.
The files to be used are placed in "sample/reaction", if the directory path is not specified.

4.6.1 Preparation of the input data by SILK

SILK is an auxiliary tool which creates the input file of **COGNAC**. Refer to the chapter 6 for the details. In order to use **SILK**, the input UDF file for **SILK** is required, in which the conditions and potential of a simulation are described. The input UDF data for **SILK** has the almost same structure as **COGNAC** input UDF data. The "potential_map.udf" currently prepared as a template is explained, for example.

1. Selection of the input UDF file for **SILK** ("potential_map.udf")

   First, refer to the UDF file for the input of **SILK** ("python/silk/sample/potential_map.udf"). The UDF file "potential_map.udf" consists of a set of records (unit of UDF data). A set of sample of simulation conditions and potential parameters is stored in each record. Here, the 6th record (the record number is **5** and the record label is "BS_REACTION") is used.

2. Load of the UDF file ("potential_map.udf")

   Start **GOURMET** and open UDF file "potential_map.udf". Since the data of the record number **0** is displayed on the window, move the slide bar under the **GOURMET** window right until the record label "BS_REACTION" is displayed on a lower right window.

3. Definition of a set of molecules (Loading, editing and execution of Python **SILK** script)

   Here, a procedure is explained to model the system consisting of two kinds of 100 monoatomic molecules.

   (a) Load of Python **SILK** script
   Click a button **Python:Load...** in the lower part of **GOURMET** window and load "python/silk/sample/silk_use_reaction.py" from the appeared file browser.

   (b) Edit of the Python **SILK** script
   Users should edit the section (**SECTION "USER DEFINITION"**) in the loaded script.

   - Output location
     The contents of **SUBSECTION "outputpath"** in **"USER DEFINITION"** are explained.

     * If # is placed at the beginning of a line in Python, the line becomes a comment.

```
####### SUBSECTION "outputpath" ######
def setOutParam(self):
#output Directory (ex. outDir="c:/OCTA8.3/***" (dos), outDir="/home/yourdir/***")
self.engine.outDir="C:/OCTA8.3/ENGINES/COGNAC/python/silk/sample"
#filename without suffix(.udf)
self.engine.cognacFileName="react3_in"
#project name
self.engine.prjName="DEVELOP"
#output file is diveded to Structure_data and other Parameters when "TWO_FILES" is cho
#self.engine.fileNumCom="ONE_FILE"
self.engine.fileNumCom="TWO_FILES"
```

The directory name, where the output file is created, is specified in **self.engine.outDir**.

The output file name is specified in **self.engine.cognacFileName** without the file extension ".udf". In this example, "react3_in.udf" is created as **COGNAC** input UDF.

The project name of output file is specified in **self.engine.prjName**.

The flag of whether to create one file or two files for the output of **SILK** (input of **COGNAC**) is specified in **self.engine.fileNumCom**. When **self.engine.fileNumCom** is specified as **ONE_FILE**, the input data for **COGNAC** is output into a single file .
When **self.engine.fileNumCom** is specified as **TWO_FILE**, the input data for **COGNAC** is output into two files. One file has a topological data and the other has the rest of input data.

- Modeling of molecule (1)
  The contents of **SUBSECTION "system"** in **"USER DEFINITION"** are explained.
  The part of the **SILK** script, which defines molecule **beadC**, is as follows.

```
##### BuildSytem(Make sure to execute at "record 5") #####
self.engine.createMolecule("beadC")
self.engine.addAtoms("beadC", "C", "C")
self.engine.addInteractionSites("beadC", [0], "siteType1", "PAIR")
self.engine.setSystem("beadC", 60)
#
```

  Each function and argument is explained below.

  i. Registration of the name of molecules:

```
self.engine.createMolecule("beadC")
```

  beadC is a name of the molecule.

  ii. Registration of atom to the registered molecule (name of the molecule : **beadC**)

```
self.engine.addAtoms("beadC", "C", "C")
```

  An atom is registered to the molecule **beadC**.
  Since the molecule **beadC** is modeled as a monoatomic molecule, this function is performed at once (only one atom is registered.)

iii. Registration of interaction site to the registered molecule (name of the molecule : **beadC**)

```
self.engine.addInteractionSites("beadC",[0],"siteType1","PAIR")
```

The interaction sites are registered for non-bonding interaction. The function **addInteractionSites(...)** is used to register the interaction sites in **SILK**.
An argument [**0**] is an array of atom. Since the interaction sites can be defined by two or more atoms in **COGNAC**, the atom(s) are specified by an array. And, **SILK** refers to the data, which has **Name siteType1_siteType1**, in the array **Interaction.Pair_Interactions**[] (**Interaction.Pair_Interactions**[**0**] in this case) for non-bonding interaction between the interaction sites.

iv. Specification of the molecules for the output from the registered molecules (name of the molecule : **beadC**)

```
self.engine.setSystem("beadC",60)
```

Here, the number of molecules to be created from the registered molecule **beadC** is specified. Notice that the data is not output unless the function **setSystem(...)** is called even if the molecular data is registered to **SILK**.

- Modeling of molecule (2)

```
#
self.engine.createMolecule("beadC2")
self.engine.addAtoms("beadC2", "C2", "C2")
self.engine.addInteractionSites("beadC2", [0], "siteType1", "PAIR")
self.engine.setSystem("beadC2", 40)
```

Here, the monoatomic molecule **beadC2** is defined. Since the procedure is completely the same as that for **beadC**, the details of the functions is omitted.

(c) Execution of the Python **SILK** script
After the contents (output place etc.) of the edited Python script are confirmed, execute the script. A result of execution will be displayed in a log window.

4.6.2 Edit of input data, and execution of COGNAC

Load the data created by **SILK** into **GOURMET**, then after the content is confirmed, execute **COGNAC**.

1. Load of the **COGNAC** input UDF created by **SILK**

   From **GOURMET Editor** window, select **File** → **Open...** and choose "react3_in.udf", which is created in the previous section, from the appeared file browser.

2. Confirmation of potential

   The left-hand side of the window, tree-like data structure is displayed. The data of the present record can be shown or edited by clicking this data structure.

   The bond potentials added by the bond creation under the simulation are registered into the UDF path **Molecular_Attributes.Bond_Potential**[].

3. Setting of the initial structure

   The information about initial structure is set up in the UDF path **Initial_Structure**. Here, random structures are generated with the density **0.6**.

4. Setting of the calculation conditions

   The calculation conditions are set up by editing the UDF Path **Simulation_Conditions**. The setup parameters are as follows.

   (a) Dynamics condition
   The data in the UDF path **Simulation_Conditions.Dynamics_Conditions** is explained.

   Time.delta_T (setting value : **0.001**) ··· Time of each dynamics step
   Time.Total_Steps (setting value : **10000**) ··· Total time steps of the simulation
   Time.Output_Interval_Steps (setting value : **20**) ··· Interval of output
   Temperature.Temperature (setting value : **5.0**) ··· Target temperature
   Temperature.Interval_of_Scale_Temp (setting value : **10 millions**) ··· Number of interval steps of velocity scaling

   * Velocity scaling is not performed when **Temperature.Interval_of_Scale_Temp** is set larger than **Time.Total_Steps**. Here, since temperature control is carried out by other technique, there is no necessity of the velocity scaling.

   (b) Solver
   The data in the UDF path **Simulation_Conditions.Solver** is explained.

   Solver_Type (setting value : **Dynamics**) ··· Selection of dynamics (**Dynamics**) or minimization (**Minimize**)
   Here, **Dynamics** is selected.

   Dynamics.Dynamics_Algorithm (setting value : **NVT_Kremer_Grest**) ··· Algorithm of dynamics
   Here, the Langevin dynamics is selected.

   Dynamics.NVT_Kremer_Grest.Friction (setting value : **0.5**) ··· Friction constant for the Langevin dynamics.

   (c) Boundary conditions
   The data in the UDF path **Simulation_Conditions.Boundary_Conditions** is explained.

   Boundary_Conditions (setting value : **PERIODIC** in all axes) ··· Boundary condition in each axis of a unit cell

   3D periodic boundary conditions are set by setting all of **a_axis**, **b_axis**, and **c_axis** to **PERIODIC**.

   (d) Flags for calculating potential terms
   The data in the UDF path **Simulation_Conditions.Calc_Potential_Flags** is explained.

   Bond (setting value : **1**)
   Angle (setting value : **0**)
   Torsion (setting value : **0**)
   Non_Bonding (setting value : **1**) ··· Setting of the calculation conditions of the bonding, non-bonding and external interaction

   When **1** is set in each item, the flag is turned on, and when **0** is set, it is turned off. Here, bond stretching (**Bond**) and non-bonding interaction (**Non_Bonding**) are turned on.

(e) Setting of the output data item
The data in the UDF path **Simulation_Conditions.Output_Flags** is explained.

Structure.Position (setting value : **1**)
Structure.Velocity (setting value : **1**)
Structure.Force (setting value : **0**) $\cdots$ Selection of the output about the position, velocity, and force of each atom
Here, position and velocity of atoms are output.

5. Save of the UDF

After each parameter is edited, the UDF file should be updated from the menu, **File** $\rightarrow$ **Save** in **GOURMET**.

6. Execution of **COGNAC**

Start from the **GOURMET Engine Run** command with the procedure shown in the chapter 3, or using a command prompt/shell window, move to the directory where UDF files are placed, and execute the following command:

```
cognac92 -I react3_in.udf -O react3_out.udf > react3.log
```

After the simulation is completed, "react3_out.udf" (UDF output), "react3_out.dat" (data output), and "react3.log" (log file) are created.

4.6.3 Display and analysis of calculation results

Display of molecular structure

As it is shown in the chapter 3, molecular structure can be displayed by **Action command**.
Chemical bonds are created based on the minimum image distance between atoms. If the distance between the displayed position of atoms is not the same as that of the minimum image, the bond is not displayed even though the chemical bond is defined between the atoms. Therefore, notice that the number of bonds displayed is fewer than that actually generated.

Analysis of the change of number of molecules

Analyze the number of molecules under the simulation with reactions.

1. Load of the output UDF

In **GOURMET Editor** window, select **File** $\rightarrow$ **Open...** and select "reaction_out.udf" from the file browser.

2. Load and execution of Python script

Select **Python:Load...** and load "CountMol.py" from the appeared file browser. Then execute the script by selecting **Run** button. After the script is executed, a number of molecules as a function of time will be set in **GraphSheet**, which is shown at **Table mode**.

3. Plotting of the number of molecules as a function of time

Select **Plot** from the tag **Python/Plot** which is in the lower left of the window after **GraphSheet** is selected in the UDF tree of the screen. Then select **Plot:Make** in the plot menu. The command line of gnuplot appears in a window. If the plot command is executed using the command line, the index of **GraphSheet** will become a horizontal axis. Therefore, rewrite the command line as follows.

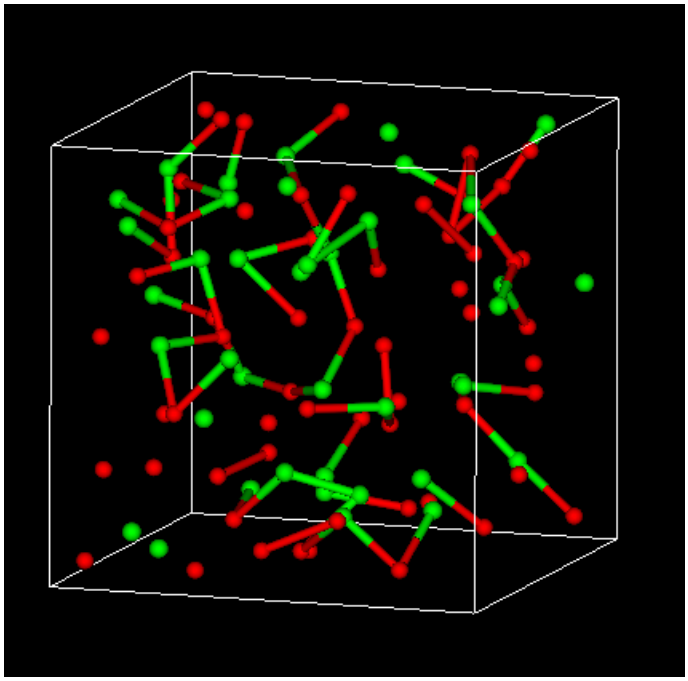

Figure 4.17: Display of molecules with **boundary_condition='atom'**

```
(original)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 1:2 title 'Time' with lines , \
     "plot.dat" using 1:3 title 'NumofMol' with lines

(after modification)
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 2:3 title 'NumofMol' with lines
```

When **Plot:Plot** is selected after that, a number of molecules as a function of time is plotted as shown in Figure 4.18.

4.7 (Example VI) Simulation of semi-crystalline lamella

This section shows an example of modeling semi-crystalline lamellae using the lamella generator. The configurations of amorphous part of chains between lamellae reproduce the results of a mean field theory.

The files to be used are placed in "sample/lamella", if the directory path is not specified.

4.7.1 Preparation of the input data by SILK

SILK is an auxiliary tool which creates the input file of **COGNAC**. Refer to the chapter 6 for the details. In order to use **SILK**, the input UDF file for **SILK** is required, in which the conditions and potential of a simulation are described. The input UDF data for **SILK** has the almost same structure as **COGNAC**

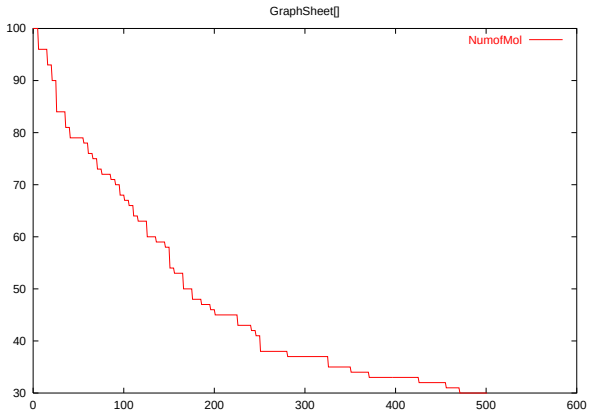

Figure 4.18: Number of molecules as a function of time

input UDF data. The "potential_map.udf" currently prepared as a template is explained, for example.

1. Selection of the input UDF file for **SILK** ("potential_map.udf")

   First, refer to the UDF file for the input of **SILK** ("python/silk/sample/potential_map.udf"). The UDF file "potential_map.udf" consists of a set of records (unit of UDF data). A set of sample of simulation conditions and potential parameters is stored in each record. Here, the 5th record (the record number is **4** and the record label is "BS_LAMELLA") is used.

2. Load of the UDF file ("potential_map.udf")

   Start **GOURMET** and open UDF "potential_map.udf". Since the data of the record number **0** is displayed on the window, move the slide bar under the **GOURMET** window right until the record label "BS_LAMELLA" is displayed on a lower right window.

3. Definition of a set of molecules (Loading, editing and execution of Python **SILK** script)

   Here, a procedure is explained to model the system consisting of 4 bead-spring type chains. This system also can be created by the command **SILK_CREATE_LinearPolymer_Bead_Spring_1_Home** of **Action SILK**. See chaper 6 fot the detail.

   (a) Load of Python **SILK** script
   Click a button **Python:Load...** in the lower part of **GOURMET** window and load "python/silk/sample/silk_use_lamella.py" from the appeared file browser.

   (b) Edit of the Python **SILK** script
   Users should edit the section (**SECTION "USER DEFINITION"**) in the loaded script.

   - Output location
     The contents of **SUBSECTION "outputpath"** in **"USER DEFINITION"** are explained.

     * If # is placed at the beginning of a line in Python, the line becomes a comment.

```
####### SUBSECTION "outputpath" ######
def setOutParam(self):
#output Directory (ex. outDir="c:/OCTA8.3/***" (dos), outDir="/home/yourdir/***")
```

```
self.engine.outDir="C:/OCTA8.3/ENGINES/COGNAC/python/silk/sample"
#filename without suffix(.udf)
self.engine.cognacFileName="bs_lamella_in"
#project name
self.engine.prjName="DEVELOP"
#output file is diveded to Structure_data and other Parameters when "TWO_FILES" is cho
#self.engine.fileNumCom="ONE_FILE"
self.engine.fileNumCom="TWO_FILES"
```

The directory name, where the output file is created, is specified in **self.engine.outDir**.

The output file name is specified in **self.engine.cognacFileName** without the file extension ".udf". In this example, "bs_lamella_in.udf" is created as **COGNAC** input UDF.

The project name of output file is specified in **self.engine.prjName**.

The flag of whether to create one file or two files for the output of **SILK** (input of **COGNAC**) is specified in **self.engine.fileNumCom**. When **self.engine.fileNumCom** is specified as **ONE_FILE**, the input data for **COGNAC** is output into a single file .
When **self.engine.fileNumCom** is specified as **TWO_FILE**, the input data for **COGNAC** is output into two files. One file has a topological data and the other has the rest of input data.

- Modeling of molecule
  The contents of **SUBSECTION "system"** in **"USER DEFINITION"** are explained.
  Here, template function **makeLinPolym(...)** of **SILK** is used.

```
##### BuildSytem(Make sure to execute at "record 4") #####
name="BS_lamella"
numAtom=1200
numMol=4
atomName= "Name1"
atomTypeName="atom1"
bondTypeName="bond1"
angleTypeName=""
torsionTypeName=""
interactionSiteTypeName="siteType1"
self.engine.makeLinPolym(name, numAtom, numMol,  bondTypeName,
                         atomName, atomTypeName,angleTypeName,
                         torsionTypeName, interactionSiteTypeName)
#
```

The function **makeLinPolym(...)** in the lowest line is a function which creates linear chain molecules.
The **self.engine.** is an indispensable description in order to use this function.

The argument of the function **makeLinPolym(...)** is explained below.

The **name** is a name of molecules.

The **numAtom** is a number of atoms in a molecule.

The **numMol** is a number of molecules.

The **atomName** is a name of each atom in a molecule.

The **atomTypeName** is a name of atom type, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Atom_Type**[].

The **bondTypeName** is a name of bond potential, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Bond_Potential**[].

The **angleTypeName** is a name of angle potential, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Angle_Potential**[].

The **torsionTypeName** is a name of torsion potential, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Torsion_Potential**[].

The **interactionSiteTypeName** is a name of interaction site type, and it corresponds to the **Name** in the UDF data **Molecular_Attributes.Interaction_Site_Type**[].
Since this molecule uses bead-spring model, the **angleTypeName** and the **torsionTypeName** are kept blank.

(c) Execution of Python **SILK** script
After the contents (output place etc.) of the edited Python script are confirmed, execute the script. A result of execution will be displayed in a log window.

4.7.2 Edit of input data, and execution of COGNAC

Load the data created by **SILK** into **GOURMET**, then after confirming the content, execute **COGNAC**.

1. Load of the **COGNAC** input UDF created by **SILK**

   From **GOURMET Editor** window, select **File** → **Open...** and choose "bs_lamella_in.udf", which is created in the previous section, from the appeared file browser.

2. Confirmation of potential

   The left-hand side of the window, tree-like data structure is displayed. The data of the present record can be shown or edited by clicking this data structure.

3. Setting of the initial structure

   The information about initial structure is set up in the UDF path **Initial_Structure**.
   Here, select **Lamella** in **Initial_Structure.Generate_Method.Method**, and
   set the parameters in **Initial_Structure.Generate_Method.Lamella**. Refer to the section 5.2.3 for the detail of the description of parameters.
   Next, confirm **Initial_Structure.Initial_Unit_Cell.Cell_Size** is set as $\mathbf{a} = \mathbf{b} = \mathbf{0.0}$ and $\mathbf{c} = \mathbf{20.0}$.
   In the case of using the lamella generator, the densities of crystalline phase and amorphous phase are specified independently, and the value of **Initial_Structure.Initial_Unit_Cell** is ignored
   Open **Initial_Structure.Generate_Method.Lamella** and confirm that **Lamella_Length** is set as **15.0**. Thereby, a crystalline phase is generated in the range of $0.0 - 15.0$ in $z-$axis, while the unit cell length of $z-$axis is **20.0**. In addition, confirm the contents of
   Initial_Structure.Generate_Method.Lamella.Random_Param
   and **Initial_Structure.Generate_Method.Lamella.Helix_Param**.

4. Setting of the calculation conditions

   The calculation conditions are set up in the UDF path **Simulation_Conditions**. The setup parameters are as follows.

(a) Dynamics condition
The data in the UDF path **Simulation_Conditions.Dynamics_Conditions** is explained.

Time.delta_T (setting value : **0.01**) ··· Time of each dynamics step
Time.Total_Steps(setting value : **10000**) ··· Total time steps of the simulation
Time.Output_Interval_Steps (setting value : **1000**) ··· Interval of output
Temperature.Temperature (setting value : **0.1**) ··· Target temperature
Temperature.Interval_of_Scale_Temp (setting value : **100**) ··· Number of interval steps of velocity scaling

(b) Solver
The data in the UDF path **Simulation_Conditions.Solver** is explained.

Solver_Type (setting value : **Dynamics**) ··· Selection of dynamics (**Dynamics**) or minimization (**Minimize**)
Here, **Dynamics** is selected.

Dynamics.Dynamics_Algorithm (setting value : **NVE**) ··· Algorithm of dynamics

* There is no additional parameter in algorithm **NVE**.

(c) Boundary conditions
The data in the UDF path **Simulation_Conditions.Boundary_Conditions** is explained.

Boundary_Conditions (setting value : **PERIODIC** in all axes) ··· Boundary condition in each axis of a unit cell
3D periodic boundary conditions are set by setting all of **a_axis**, **b_axis**, and **c_axis** to **PERIODIC**.

(d) Flags for calculating potential term
The data in the UDF path **Simulation_Conditions.Calc_Potential_Flags** is explained.

Bond (setting value : **1**)
Angle (setting value : **0**)
Torsion (setting value : **0**)
Non_Bonding (setting value : **1**) ··· Setting of the calculation conditions of the bonding, non-bonding and external interaction.

When **1** is set in each item, the flag is turned on, and when **0** is set, it is turned off. Here, bond stretching (**Bond**) and non-bonding interaction (**Non_Bonding**) are turned on.

(e) Setting of the output data item
The data in the UDF path **Simulation_Conditions.Output_Flags** is explained.

Structure.Position (setting value : **1**)
Structure.Velocity (setting value : **0**)
Structure.Force (setting value : **0**) ··· Selection of the output about the position, velocity, and force of each atom

Here, only the position of atoms is output.

5. Save of the UDF

   After each parameter is edited, the UDF file should be updated from the menu, **File** → **Save** in **GOURMET**.

6. Execution of **COGNAC**

   Start from the **GOURMET Engine Run** command with a procedure shown in the chapter 3, or using the command prompt/shell window, move to the directory where UDF files are places, and execute the following command:

   ```
   cognac92 -I bs_lamella_in.udf -O bs_lamella_out.udf > bs_lamella.log
   ```

   After the simulation is completed, "bs_lamella_out.udf" (UDF output), "bs_lamella_out.dat" (data output) and "bs_lamella.log" (log file) are created.

4.7.3 Display and analysis of calculation results

Display of molecular structure

As it is shown in the chapter 3, molecular structure can be displayed by **Action command**.
If the argument **color** is set as **'mol'**, molecules are displayed with different color by molecule as shown in Figure 4.19.

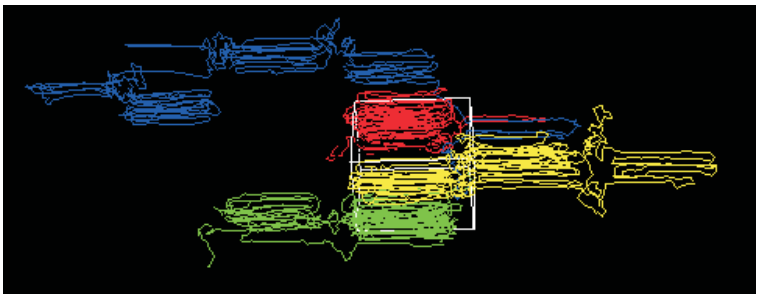

Figure 4.19: Display of molecules with **color='mol'**

Export the molecular structure to Accelrys car format file

Export the molecular topology and coordinate data of UDF to a **car** format file of Accelrys Inc. A lamella structure of CH_2 united atom created by the same method as that of the bead-spring model is used, for example. **Action command : EXPORT_data...** can be used to execute the same procedure. See details in section 7.5.

1. Load of the output UDF
   In **GOURMET Editor** window, select **File** → **Open...** and choose "out/ua_lamella_out.udf" from the appeared file browser.

2. Load and execution of Python script
   Move to arbitrary record to convert the data, then select **Python:Load...** and load "FileConvert.py" from the appeared file browser. After the script is executed, a file "ua_lamella_out.car" is created in the directory, where the loaded UDF exists.
   The scale parameter in the script corresponds the length of 1σ in the unit Å.

Created **car** file can be read with products of Accelrys Inc. such as Material Studio™. However, since **mdf** files which have an information of bond connection is not created, bonds may not be displayed correctly depending on the software.
If the hydrogen atoms are added to the data read by the software of Accelrys Inc. and potentials are assigned, atomistic simulations can be performed using the software of Accelrys Inc. such as Discover®program.

4.8 (Example VII) Simulation of microphase separation of block copolymer by DPD

This section shows an example of the simulation of microphase separation of AB diblock copolymer by DPD. The files to be used are placed in "sample/DPD", if the directory path is not specified.

4.8.1 Preparation of the input data by Action SILK

This section explains how to define the topological data of 2,400 molecules of A5B5 diblock copolymer by **Action SILK**

1. Selection of the UDF file for the input of **Action SILK** ("potential_map.udf")

   First, refer to the UDF file for the input of **Action SILK** ("python/silk/sample/potential_map.udf"). Here, the 10th record (the record number is **9** and the record label is "DPD") is used.

2. Load of the UDF file ("potential_map.udf")

   Start **GOURMET** and open UDF file "potential_map.udf" from the menu **File** → **Open...**. Since the data of the record number **0** is displayed on the window, move the slide bar under the **GOURMET** window right until the record label "DPD" is displayed on a lower right window.

3. Definition of a set of molecules

   (a) Selection of the template command of **Action SILK**
   Click the folder like icon of **Set_of_Molecules** in the loaded "potential_map.udf" with right button of mouse, then some **Action SILK** command will be appeared. (Since **Set_of_Molecules** is empty at initial state, the colored of icon is gray.)
   Select command **SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block...** from the appeared commands to model bead-spring type diblock copolymer.

   (b) Setting of **SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block...**
   Here, 2,400 molecules of A5B5 diblock copolymer are defined.
   When **SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block...** is selected, a new window will be appeared. The parameters for modeling diblock copolymer are specified as it is shown in Figure 4.20.
   See the detail of parameters in Chapter 3 and 6.
   After all parameters are specified, select **OK**. The information of created molecules are not put in **Set_of_Molecules** yet.

4. Output to **Set_of_Molecules**

   Since only one type of moelcule is defined in this example, put the created information of molecules to **Set_of_Molecules** now.

   Click the folder like icon of **Set_of_Molecules** in the loaded "potential_map.udf" with right button of mouse again, then select **SILK_OUT_SYSTEM_to_Set_of_Molecules...** from the command list. When selecting **OK** in the appeared window with keeping parameter **Continue**, **Set_of_Molecules** is updated in the current record of "potential_map.udf". Notice that the data of **Set_of_Molecules** is only exist in the cache memory of **GOURMET**, and "potential_map.udf" file is not update until

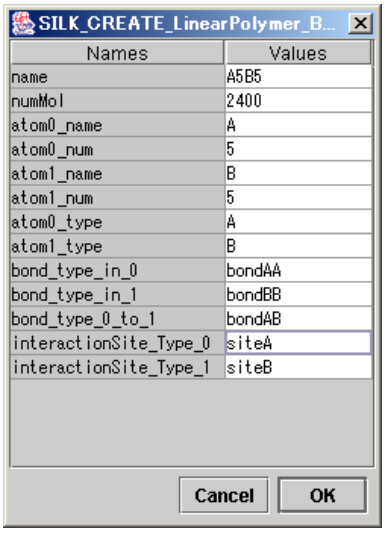

Figure 4.20: Setting parameters for **SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block...**

you save the UDF in **GOURMET**. It is better not to overwrite the template "potential_map.udf".

5. Create a new UDF file

   Create a new UDF file from the record of "potential_map.udf", where defined **Set_of_Molecules** and template parameter for calculation conditions.
   To do this, click the root icon of "potential_map.udf" with right button of mouse, then select **SILK_UDF_CREATE...** from the appeared command list. When specifying the file name in the appeared window and select **OK**, the input UDF file for **COGNAC** is created. The file name "A5B5_in.udf" is used for the following example.

4.8.2 Edit of input data, and execution of COGNAC

Load the UDF file created by **Action SILK** into **GOURMET**, then after editing of the content, execute **COGNAC**.

1. Load of the **COGNAC** input UDF created by **SILK**

   From **GOURMET Editor** window, select **File** → **Open...** and choose "A5B5_in.udf", which is created in the previous section, from the appeared file browser.

2. Setting of the initial structure

   The information about initial structure is set up in the UDF path **Initial_Structure**. The main parameters of **Initial_Structure** are explained:

   Initial_Unit_Cell.Density ··· Density of the system specified as an initial condition

   The reduced unit value **3.0** is set in this example.

   Generate_Method.Method ··· Method for initial structure generation

   Random is specified in this example and the data below the UDF Path **Generate_Method.Random** is used.
   The other data (for example, UDF Path **Initial_Structure.Generate_Method.Helix** etc.) which exist in the same layer are ignored.

3. Setting of the calculation conditions

   The calculation conditions are set up by editing the UDF Path **Simulation_Conditions**. The setup parameters are as follows.

   (a) Dynamics condition
   The data in the UDF path **Simulation_Conditions.Dynamics_Conditions** is explained.

   Time.delta_T (setting value : **0.06**) ··· Time of each dynamics step
   Time.Total_Steps (setting value : **10000**) ··· Total time steps of the simulation. Since 10,000 time steps may bot not enough to obtain an equilibrium morphorgy, extending the time steps to about 100,000 is recommended. However, notice that it will take long computational time.
   Time.Output_Interval_Steps (setting value : **1000**) ··· Interval of output
   Temperature.Temperature (setting value : **1.0**) ··· Target temperature
   Temperature.Interval_of_Scale_Temp (setting value : **0**) ··· Number of interval steps of velocity scaling

   * Velocity scaling is not performed when **Temperature.Interval_of_Scale_Temp** is 0. Here, since temperature control is carried out by DPD solver itself, there is no necessity of the velocity scaling.

   (b) Solver
   The data in the UDF path **Simulation_Conditions.Solver** is explained.

   Solver_Type (setting value : **Dynamics**) ··· Selection of dynamics (**Dynamics**) or minimization (**Minimize**)
   Here, **Dynamics** is selected.

   Dynamics.Dynamics_Algorithm (setting value : **DPD**) ··· Algorithm of dynamics
   Here, the DPD is selected.

   Dynamics.DPD.lambda (setting value : **0.65**) ··· Parameter required for integrating the equation of motions of DPD. Refer to eq.2.37

   (c) Boundary conditions
   The data in the UDF path **Simulation_Conditions.Boundary_Conditions** is explained.

   Boundary_Conditions (setting value : **PERIODIC** in all axes) ··· Boundary condition in each axis of a unit cell
   3D periodic boundary conditions are set by setting all of **a_axis**, **b_axis**, and **c_axis** to **PERIODIC**.

   (d) Flags for calculating potential terms
   The data in the UDF path **Simulation_Conditions.Calc_Potential_Flags** is explained.

   Bond (setting value : **1**)
   Angle (setting value : **0**)
   Torsion (setting value : **0**)
   Non_Bonding (setting value : **1**) ··· Setting of the calculation conditions of the bonding, non-bonding and external interactions

   When **1** is set in each item, the flag is turned on, and when **0** is set, it is turned off. In the case of usual DPD, only bond stretching (**Bond**) and non-bonding interaction (**Non_Bonding**) are turned on.

(e) Setting of the output data item
The data in the UDF path **Simulation_Conditions.Output_Flags** is explained.

Statistics ··· Selection of output items, such as temperature and pressure (Selected data are output to data and log file. However, all data are output to UDF file.)
Structure.Position (setting value : **1**)
Structure.Velocity (setting value : **0**)
Structure.Force (setting value : **0**) ··· Selection of the output about the position, velocity, and force of each atom
Here, only the position of atoms is output.

4. Save of the UDF

After each parameter is edited, the UDF file should be updated from the menu, **File** → **Save** in **GOURMET**.

5. Execution of **COGNAC**

Start from the **GOURMET Engine Run** command with the procedure shown in the chapter 3, or using a command prompt/shell window, move to the directory where UDF files are placed, and execute the following command:

```
cognac92 -I A5B5_in.udf -O A5B5_out.udf > A5B5.log
```

Refer to the section 5.1 about the details of input and output files and the arguments.

After the simulation is completed, "A5B5_out.udf" (UDF output), "A5B5_out.dat" (data output), and "A5B5.log" (log file) are created.

4.8.3 Display and analysis of calculation results

Display of molecular structure

As it is shown in the chapter 3, molecular structure can be displayed by **Action command**. Figure 4.21 shows an example of display with setting **type** to **ball-stick** and **bc** to **atom**.

The lamella morphorgy was obtained by the DPD simulation.

Calculation of scattering function

Using a command **ANALYSIS_scattering_function...** in the **Action** menu, you can calculate scattering function from the distribution of atoms. A command window will be appeared by selecting the **ANALYSIS_scattering_function...** as it is shown in the Figure 4.22. When **Run** button is selected after the paramters are set as in the Figure, the scattering function will be plotted as it is shown in the Figure 4.23. See the detail of the parameter in the "Readme" file too.

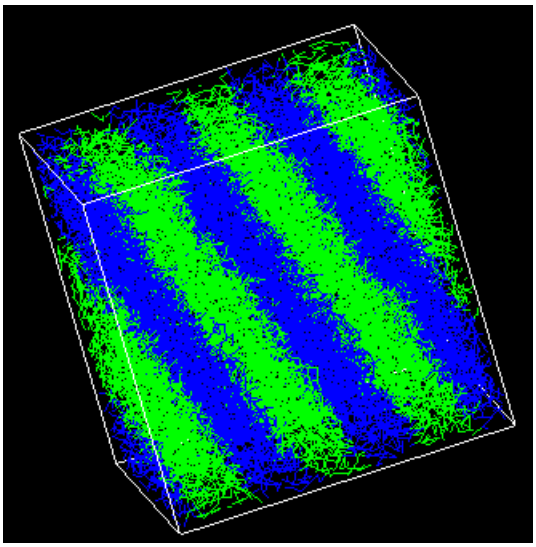

Figure 4.21: Display of molecular structure with **type='ball-stick',bc = 'atom'**

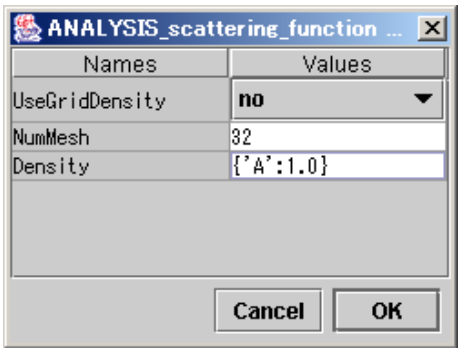

Figure 4.22: Parameter window of **ANALYSIS_scattering_function...**

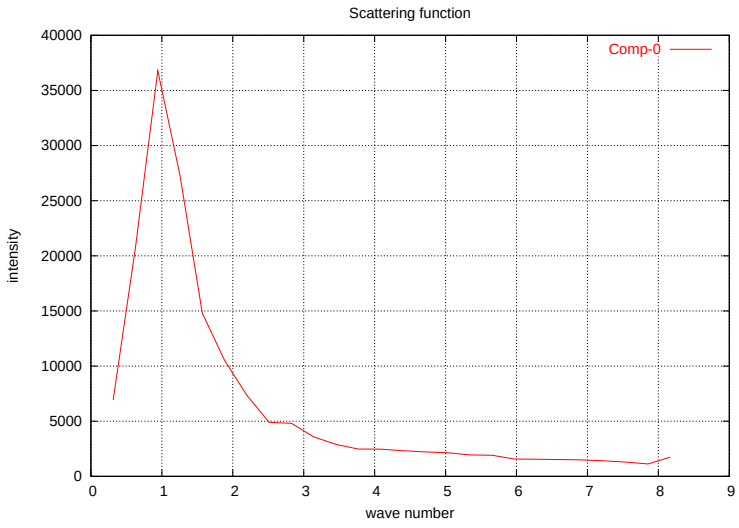

Figure 4.23: Plot of scattering function

Chapter 5

Operation guide of COGNAC

This chapter explains (i) how to execute **COGNAC**, (ii) the detailed description of UDF files and (iii) parameters used for input and output of **COGNAC**.

5.1 How to execute COGNAC

5.1.1 List of UDF

List of **COGNAC** UDF

- Definition UDF ("cognac92.udf") ··· This file defines data structures of **COGNAC** input/output. Usually, it is installed automatically to the proper directory at the installation of OCTA.
- Input UDF ··· Input data for **COGNAC**
- Output UDF ··· Output data for **COGNAC**
- Table UDF ··· Data for table potential
- Crystal UDF ··· Data for crystal generator, i.e. cell size, symmetrical operation and fractional coordinates

5.1.2 Execution of COGNAC on GOURMET

After input the following arguments in **Engine Run panel** of **GOURMET** window, start **COGNAC** with **RUN** button.

- **Run name**: ··· Selection of registered run name
- **Server**: ··· Server host name. Keep blank if executing **COGNAC** at local host
- **Engine**: ··· Automatically set when selecting proper **Run name**
- **Working Dir**: ··· Working directory to execute **COGNAC**.
  Specify arbitrary directory on server except the directory where input UDF exists.
- **Parms**: ··· Usually, not used in **COGNAC**. Keep blank.
- **Input UDF**: ··· Input UDF file name
  It is specified automatically if **Always Use Current UDF at Input UDF** is checked.
- **Params UDF**: ··· Not used in **COGNAC**. Keep blank
- **Restart UDF**: ··· Restart UDF file name
  When restart UDF is used at the execution of **COGNAC**, specify UDF file name for restart here.
- **Output UDF**: ··· Output UDF file name

- **Summary UDF**: ··· Summary UDF file name to be used at the **Engine Control panel** Arbitrary directory and file name can be specified.
- **Logger**: ··· Not used in **COGNAC**. Keep blank.

5.1.3 Execution of COGNAC on command/shell window

When executing **COGNAC** from command/shell window, proper environmental variables must be set beforehand. A command **gourmetterm** (placed in "GOURMET/bin") will open command/shell window with the environmental variables are set.

% cognac92 -I *input UDF* {-O *output UDF*} {-R *restart UDF*} {-n *number of thread*} {-p *order of precision of Structure output*} {-s *random seed*}

- input UDF ··· Input UDF file
- output UDF (optional) ··· Output UDF file
- restart UDF (optional) ··· Restart UDF file
- number of thread (optional) · Number of thread at parallel run. Default is 1
- order of precision of Structure output ··· Order of precision of Position, Velocity and Force in trajectory output. Default value is 6. However, the precision of the last record is fixed to 14 for restart.
- random seed ··· Random seed. If not specified, program will set the value automatically.

  NOTE: OpenMP executable file made in Win/MinGW environment needs a file pthreadGC2.dll. You can download the file, and put the file in the directory where cognac executable exists.

1. When the output UDF file is not specified and the restart UDF file is specified by **-R**, the output is appended to restart UDF file.
2. When both the output UDF file and the restart UDF file are not specified, the output is appended to the input UDF file.
3. When the output UDF is specified as the same name as the restart UDF, the output is appended to the restart UDF file.
4. When the restart UDF file is specified with **-R**, it overwrite the restart UDF name specified in the input UDF file, and **Set_of_Molecules** data will be read from the UDF specified with **-R**. And in the case **Initial_Structure.Generate_Method.Method** is **"RESTART"** , the **Structure** data will be also read from the UDF file.

5.1.4 Miscellaneous arguments for COGNAC

COGNAC takes other argumets without executing the program.

- -V ··· Display version of executable
- -C ··· Convertion of old version Input UDF to current version.
  Example
  %cognac92 -I*Filename of old version Input UDF* -O*Created file name of current version Input UDF* -C

5.1.5 Termination of job

When **COGNAC** is started from **GOURMET**, the job can be terminated by **Engine Control panel** .
If **Stop** is selected in **Engine Control panel** , **COGNAC** will output the information of the current step and terminate normally.
If **Kill** is selected, the process is killed and the content of output UDF file is not guaranteed.

5.2 Description of input UDF

Input UDF consists of the following maximum eight objects.

- **Simulation_Conditions**

  Setup of the general simulation condition, e.g. temperature, pressure, time step and ensemble

- **Initial_Structure**

  Setup of the initial structure, e.g. random, helix, crystal and restart

- **Molecular_Attributes**

  Setup of the atomic type, interaction site and bonding potential

- **Interactions**

  Setup of the non-bonding interaction, Electrostatic potential and external potential

- **Set_of_Molecules**

  The definition of molecular architecture, i.e. topology, potential assignment
  This object can be read from a separated file.

- **Structure**

  The information of molecular structure, i.e. coordinates, velocity and force, and the unit cell
  This object can be read from a separated file, and it is not necessary, when the initial coordinate is generated in **COGNAC**.

- **React_Conditions**

  Setup for dealing with a chemical reaction (creation and scission of bonds)
  This is not necessary in the simulation which does not include chemical reactions.

- **Unit_Parameter**

  Scaling parameters for unit conversion

- **Draw_Attributes**

  Attributes of drawing on **View** window. This is not necessary in the simulation of **COGNAC**, and the parameters in input UDF will be output to the output UDF.

Lists of parameters are shown in Table 5.1-5.8. Refer to the text for the details of each parameters.

Table 5.1: Simulation_Conditions

UDF path	Description
"Dynamics_Conditions"	Basic conditions of MD
"Dynamics_Conditions.Max_Force"	Maximum force allow during MD
"Dynamics_Conditions.Time"	Setup for time, i.e. total time step, time step etc.
"Dynamics_Conditions.Temperature"	Setup for temperature control
"Dynamics_Conditions.Pressure_Stress"	Setup for pressure and stress control
"Dynamics_Conditions.Deformation"	Setup for deformation
"Dynamics_Conditions.Moment"	Setup for moment calculation
"Dynamics_Conditions.RATTLE"	Setup for RATTLE(bond and angle constraint)
"Solver"	Setup for MD/MM
"Solver.Solver_Type"	[Keyword] **DYNAMICS/MINIMIZE**
"Solver.Dynamics"	Setup for MD
"Solver.Minimize"	Setup for MM
"Boundary_Conditions"	Boundary conditions
"Boundary_Conditions.a_axis"	Boundary conditions of a-axis
"Boundary_Conditions.b_axis"	Boundary conditions of b-axis
"Boundary_Conditions.c_axis"	Boundary conditions of c-axis
"Boundary_Conditions.Periodic_Bond"	Flag for periodic bond
"Calc_Potential_Flags"	Flags for calculating potential functions
"Calc_Potential_Flags.Bond"	Flag for bond potential on/off
"Calc_Potential_Flags.Angle"	Flag for angle potential on/off
"Calc_Potential_Flags.Torsion"	Flag for torsion potential on/off
"Calc_Potential_Flags.Non_Bonding_Interchain"	Flag for inter-molecular non-bonding
"Calc_Potential_Flags.Non_Bonding_Intrachain"	Flag for intra-molecular non-bonding
"Calc_Potential_Flags.Non_Bonding_1_3"	Flag for 1-3 non-bonding interaction on/off
"Calc_Potential_Flags.Non_Bonding_1_4"	Flag for 1-4 non-bonding interaction on/off
"Calc_Potential_Flags.External"	Flag for external interaction on/off
"Calc_Potential_Flags.Electrostatic"	Flag for electrostatic interaction on/off
"Calc_Potential_Flags.Tail_Correction"	Flag for tail correction on/off
"Output_Flags"	Flags for output
"Output_Flags.Statistics"	Flag for output of statistical data
"Output_Flags.Structure"	Flag for output of position, velocity and force
"Output_Flags.Averaged_Structure"	Flag for output of batch averaged position, velocity and force
"Output_Flags.Correlation_Function"	On the fly output of correlation functions
"Density_Output"	Setup for calculating density distribution
"Density_Output.Calc_Grid_Density"	Flag for calculating density distribution
"Density_Output.Interval_of_Calc_Density"	Interval of MD steps to collect data for density distribution
"Density_Output.Collect_Density_Steps"	Number of MD steps to collect data for density distribution
"Density_Output.Num_of_Grid"	Size of lattice
"Density_Output.Radius"	Effective radius
"Density_Output.Normalize_On"	Flag for normalization
"Constraint_Conditions"	Conditions for constraint atom
"Read_from_Restart"	Flag for read constraint conditions from restart file
"Constraint_Atom[]"	Array of constraint atoms

Table 5.2: Initial_Structure

UDF path	Description
"Initial_Unit_Cell"	Setup for unit cell
"Initial_Unit_Cell.Density"	Density
"Initial_Unit_Cell.Cell_Size"	Cell size
"Initial_Unit_Cell.Shear_Strain"	Initial shear strain at Lees-Edwards boundary condition
"Read_Set_of_Molecules"	Information of **Set_of_Molecules**
"Read_Set_of_Molecules.UDF_Name"	UDF file name of **Set_of_Molecules**
"Read_Set_of_Molecules.Record"	Record no.
"Generate_Method"	Method for generating initial structures
"Generate_Method.Method"	[Keyword] Selection of method
"Generate_Method.Restart"	Setup for restart
"Generate_Method.Random"	Setup for random structure
"Generate_Method.Helix"	Setup for helix
"Generate_Method.Lamella"	Setup for semi-crystalline lamella
"Generate_Method.Crystal"	Setup for crystal
"Relaxation"	Relaxation of initial structure
"Relaxation.Relaxation"	Setup for relaxation
"Relaxation.Method"	Method [Keyword] **DYNAMICS/MINIMIZE**
"Relaxation.Max_Relax_Force"	Maximum force at relaxation
"Relaxation.Max_Relax_Steps"	Maximum steps at relaxation

Table 5.3: Molecular_Attributes

UDF path	Description
"Atom_Type[]"	Information of atom type
"Atom_Type[*].Name"	Name of atom type
"Atom_Type[*].Mass"	Mass of atom type
"Bond_Potential[]"	Information of bond potential
"Bond_Potential[*].Name"	Name of bond potential
"Bond_Potential[*].Potential_Type"	Type of bond potential [Keyword]
"Bond_Potential[*].R0"	Equilibrium length
"Bond_Potential[*].Harmonic"	Setup for harmonic potential
"Bond_Potential[*].FENE_LJ"	Setup for FENE+LJ potential
"Bond_Potential[*].Gauss"	Setup for Gauss potential
"Bond_Potential[*].Morse"	Setup for Morse potential
"Bond_Potential[*].Bond_Polynomial"	Setup for polynomial potential
"Bond_Potential[*].DPD"	Setup for harmonic potential used in DPD simulation
"Bond_Potential[*].Table_Bond"	Setup for table type potential
"Bond_Potential[*].User_Bond"	Setup for user defined potential
"Angle_Potential[]"	Information of angle potential
"Angle_Potential[*].Name"	Name of angle potential
"Angle_Potential[*].Potential_Type"	Type of angle potential [Keyword]
"Angle_Potential[*].theta0"	Equilibrium angle
"Angle_Potential[*].Theta"	Setup for Theta harmonic potential
"Angle_Potential[*].Theta2"	Setup for Theta harmonic 2 potential
"Angle_Potential[*].Cosine"	Setup for Cosine harmonic potential
"Angle_Potential[*].Theta_Polynomial"	Setup for polynomial potential
"Angle_Potential[*].Table_Angle"	Setup for table type potential
"Angle_Potential[*].User_Angle"	Setup for User defined potential
"Torsion_Potential[]"	Information of torsion potential
"Torsion_Potential[*].Name"	Name of torsion potential
"Torsion_Potential[*].Potential_Type"	Type of torsion potential [Keyword]
"Torsion_Potential[*].Cosine_Polynomial"	Setup for polynomial potential
"Torsion_Potential[*].Amber"	Setup for Amber type torsion potential
"Torsion_Potential[*].Dreiding"	Setup for Dreiding type torsion potential
"Torsion_Potential[*].Table_Torsion"	Setup for table type potential
"Torsion_Potential[*].User_Torsion"	Setup for User defined potential
"Interaction_Site_Type[]"	Information of interaction site type
"Interaction_Site_Type[*].Name"	Name of interaction site type
"Interaction_Site_Type[*].Num_of_Atoms"	Number of atoms defining interaction site
"Interaction_Site_Type[*].Range"	Radius of site to make neighbor list
"Interaction_Site_Type[*].Rigid"	Flag for reducing degree of freedom if interaction site consisting of more than two atoms is calculated as a rigid body.
"Electrostatic_Site_Type[]"	Information of electrostatic site type
"Electrostatic_Site_Type[*].Name"	Name of electrostatic site type
"Electrostatic_Site_Type[*].Type_Name"	Type name of electrostatic site type
"Electrostatic_Site_Type[*].Polarizability"	Polarizability of electrostatic site type

Table 5.4: Interactions

UDF path	Description
"Pair_Interaction[]"	Information of pair interaction
"Pair_Interaction[*].Name"	Name of pair interaction
"Pair_Interaction[*].Potential_Type"	Type of pair interaction [Keyword]
"Pair_Interaction[*].Site1_Name"	Name of interaction site to apply pair interaction
"Pair_Interaction[*].Site2_Name"	same above
"Pair_Interaction[*].Cutoff"	Cutoff distance
"Pair_Interaction[*].Scale_1_4_Pair"	Scale factor for 1-4 pair interction
"Pair_Interaction[*].Lennard_Jones"	Setup for Lennard-Jones pair interaction
"Pair_Interaction[*].Lennard_Jones_EV"	Setup for Lennard-Jones with excluded volume pair interaction
"Pair_Interaction[*].General_Lennard_Jones"	Setup for general Lennard-Jones pair onteraction
"Pair_Interaction[*].Gay_Berne"	Setup for Gay-Berne pair interaction
"Pair_Interaction[*].GB_LJ"	Setup for Gay-Berne – Lennard-Jones pair interaction
"Pair_Interaction[*].DPD"	Setup for a pair interaction used in DPD simulation
"Pair_Interaction[*].Morse"	Setup for Morse pair interaction
"Pair_Interaction[*].Buckingham"	Setup for Buckingham pair interaction
"Pair_Interaction[*].Table_Pair_Potential"	Setup for table type pair interaction
"Pair_Interaction[*].User_Pair_Interaction"	Setup for user defined pair interaction
"External_Interaction[]"	Information of external interaction
"External_Interaction[*].Name"	Name of external interaction
"External_Interaction[*].Potential_Type"	Type of external interaction [Keyword]
"External_Interaction[*].Site_Name"	Name of interaction site to apply external interaction
"External_Interaction[*].LJ_Wall"	Setup for Lennard-Jones type flat wall
"External_Interaction[*].LJ_Atomic_Wall"	Setup for Lennard-Jones atomic type potential
"External_Interaction[*].Static_Field"	Setup for homogeneous field
"External_Interaction[*].Density_Field"	Setup for density field
"External_Interaction[*].Density_Field"	Setup for velocity field
"External_Interaction[*].Total_Density_Constrain"	Setup for total density constrain
"External_Interaction[*].External_Angle"	Setup for external angle potential
"External_Interaction[*].External_Torsion"	Setup for external torsion potential
"External_Interaction[*].User_External_Field"	Setup for user defined external interaction
"Electrostatic_Interaction[]"	Information of electrostatic interaction
"Electrostatic_Interaction[*].Name"	Name of electrostatic interaction
"Electrostatic_Interaction[*].Algorithm"	Algorithm of calculating electrostatic interaction [Keyword]
"Electrostatic_Interaction[*].Dielectric_Constant"	Dielectric constant
"Electrostatic_Interaction[*].Scale_1_4_Pair"	Scale factor for 1-4 electrostatic interaction
"Electrostatic_Interaction[*].Cutoff_Coulomb"	Setup for cutoff algorithm
"Electrostatic_Interaction[*].Reaction_Field"	Setup for reaction field
"Electrostatic_Interaction[*].Ewald"	Setup for Ewald
"Electrostatic_Interaction[*].Field_Electrostatic"	Setup for field electrostatic algorithm
"Electrostatic_Interaction[*].PPPM"	Setup for PPPM

Table 5.5: React_Conditions

UDF path	Description
"React_Flag"	Flag to include reaction
"Atom_Exchange"	Information of atom type exchange
"Exchange_Type_Array[]"	Setup for exchanging atom type
"Bond_Creation"	Information of bond creation
"Bond_Creation.Reactive_Atom[]"	Setup for reactive atom
"Bond_Creation.Creation_Type_Array[]]"	Setup for creation tyep
"Bond_Creation.Potential_Assignment"	Potential attribute for new angle/torsion
"Bond_Scission"	Information of bond scission
"Bond_Scission.Bond_Scission[]"	Setup for bond scission

Table 5.6: Set_of_Molecules

UDF path	Description
"Molecule[]"	Array of molecules
"Molecule[*].Mol_Name"	Name of molecules
"Molecule[*].atom[]"	Array of atoms in a molecule
"Molecule[*].bond[]"	Array of bonds in a molecule
"Molecule[*].angle[]"	Array of angles in a molecule
"Molecule[*].torsion[]"	Array of torsions in a molecule
"Molecule[*].interaction_Site[]"	Array of interaction sites in a molecule
"Molecule[*].electrostatic_Site[]"	Array of electrostatic sites in a molecule

Table 5.7: Structure

UDF path	Description
"Position"	Position of atoms
"Position.mol[]"	Position of atoms in a molecule.
"Velocity"	Velocity of atoms
"Velocity.mol[]"	Velocity of atoms in a molecule
"Force"	Force acting atoms
"Force.mol[]"	Force acting atoms in a molecule
"Unit_Cell"	Information of unit cell
"Unit_Cell.Density"	Density
"Unit_Cell.Cell_Size"	Unit cell
"Unit_Cell.Cell_Size.a"	length a
"Unit_Cell.Cell_Size.b"	length b
"Unit_Cell.Cell_Size.c"	length c
"Unit_Cell.Cell_Size.alpha"	angle α
"Unit_Cell.Cell_Size.beta"	angle β
"Unit_Cell.Cell_Size.gamma"	angle γ
"Unit_Cell.Shear_Strain"	Shear strain at Lees-Edwards boundary conditions

Table 5.8: Unit_Parameter

UDF path	Description
"Name"	Name of unit set
"Comment"	Comment for the unit set
"Mass"	Reduced mass
"Energy"	Reduced energy
"Length"	Reduced length

Table 5.9: Draw_Attributes

UDF path	Description
"Atom_Type"	**Atom_Type** to define attributes
"Atom_Type.Name"	Name of **Atom_Type**
"Atom_Type.color"	color
"Atom_Type.trancparency"	transparency
"Atom_Type.radius"	radius
"Bond_Potential"	**Bond_Potential** to define attributes
"Bond_Potential.Name"	Name of **Bond_Potential**
"Bond_Potential.color"	color
"Bond_Potential.trancparency"	transparency
"Bond_Potential.radius"	radius
"Molecule"	Molecule to define attributes
"Molecule.Mol_Name"	Name of molecule
"Molecule.color"	color
"Molecule.trancparency"	transparency
"Molecule.radius"	radius

Hereafter, the parameters in the UDF data are explained for every structured object.

5.2.1 Notes

Type of parameter

Each input parameter has a fixed variable type, i.e. *string, float, int....* The type of parameter is displayed with the name of parameter in GOURMET. Since the variable of *bool* type is not supported in GOURMET, type *short* is used as a substitution of the type *bool* in **COGNAC**, and the value of **1** (true) and **0** (false) is must be set into the parameter of type *short*.

Treatment of character sequences for input

For example, in the case of choosing an algorithm of dynamics, reserved words selected from the list is not case sensitive. Moreover, all blank characters before and after a character sequence or between characters are removed. On the other hand, in the case that arbitrary word can be specified such as **Atom_Name**, the input is case sensitive. Also it is case sensitive when specifying other UDF file name in a UDF file.

5.2.2 Simulation_Conditions

Dynamics_Conditions

Setup for basic simulation conditions

- **Time**: Setup for total simulation step, time step, etc.
  - **delta_T** $\cdots$ Time of each dynamics step Δt
  - **Total_Steps** $\cdots$ Total time steps of the simulation
  - **Output_Interval_Steps** $\cdots$ Interval of output data output

- **Temperature**: Setup for temperature control
  It is effective in the temperature control algorithm in MD, or velocity scaling.
  - **Temperature** $\cdots$ Target temperature
  - **Interval_of_Scale_Temp** $\cdots$ Number of interval steps for velocity scaling

- **Pressure_Stress**: Setup for external pressure and stress
  - **Pressure** $\cdots$ Target pressure
  - **Stress** $\cdots$ Target stress **(xx, yy, zz, yz, zx, xy)**

- **Deformation**: Setup for cell deformation and shear flow
  - **Method** $\cdots$ Method of deformation and shear flow
    It is selected from **Lees_Edwards**/ **Cell_Deformation**.
  - **Lees_Edwards**: Setup for shear flow by Lees-Edwards boundary conditions
    - **Method** $\cdots$ Type of shear
      It is selected from **Steady**(steady shear.Changed from **Static** in old version)/**Dynamic** (dynamic shear)
    - **Steady.Shear_Rate** $\cdots$ Shear rate in the case of steady shear
      It is given by $d\gamma_{yx}/dt[\tau^{-1}]$.
    - **Dynamic.Amplitude** $\cdots$ Amplitude in the case of dynamic shear (Maximum strain)

    * **Dynamic.Frequency** $\cdots$ Frequency $[\tau^{-1}]$ in the case of dynamic shear

  - **Cell_Deformation**: Setup for cell deformation
    * **Method** $\cdots$ Method of cell deformation
      It is selected from **Cell_Deformation/ Simple_Elongation/Oscilliation**.
    * **Interval_of_Deform** $\cdots$ Interval of time step for performing cell deformation
      It is necessary to select the proper interval of deformation empirically. If this interval of deformation is taken too long, the amount of deformations will become too large, and the simulation will become unstable. On the contrary, the simulation needs longer CPU time, if interval of deformation is taken too short.
      The deformation rate given by **Deformation_Rate** or **Simple_Elongation** will not be changed, no matter how this interval is set.
    * **Deform_Atom** $\cdots$ Flag of whether to deform each atom at the time of cell deformation
    * **Deformation_Rate** $\cdots$ Tensor of deformation rate of cell
      It is defined as $d\epsilon/dt[\tau^{-1}]$.
    * **Simple_Elongation.Elongation_Rate** $\cdots$ Velocity of elongation to the direction of z-axis
      It is defined as $dL_z/dt[\sigma\tau^{-1}]$.
    * **Simple_Elongation.Poisson_Ratio** $\cdots$ Poisson's ratio at the time of elongation (**0-0.5**)
      According to the input value, the cell size of x and y direction will be changed during simple elongation.
    * **Simple_Elongation.Axis** $\cdots$ Mode of elonation. It is selected from **z** (unixial deformation on z-axis)/**xy** (equibiaxial deformation on xy plane)
    * **Oscillation.Amplitude** $\cdots$ Amplitude in the case of dynamic elongation (Maximum strain)
    * **Oscillation.Frequency** $\cdots$ Frequency $[\tau^{-1}]$ in the case of dynamic elongation
    * **Oscillation.Poisson_Ratio** $\cdots$ Poisson's ratio at the time of dynamic elongation (**0-0.5**)
      According to the input value, the cell size of x and y direction will be changed during dynamic elongation.

  Notes:
  Shear_Rate/Deformation_Rate/Elongation_Rate cannot be set simultaneously.

- **Moment**: Setup for calculation, output and stop of translational and rotational moment.
  This operation is not guaranteed when the system has constraint atoms.
  - **Interval_of_Calc_Moment** $\cdots$ Interval of time step which calculate, output and stop moments
    It is effective only at the time of **Calc_Moment** = true(**1**).
  - **Calc_Moment** $\cdots$ Flag of whether to calculate moments of system
  - **Print_Moment** $\cdots$ Flag of whether to output moments of system to standard output
    It is effective only at the time of **Calc_Moment** = true(**1**).
  - **Stop_Translation** $\cdots$ Flag of whether to stop an translational moment of system
    It is effective only at the time of **Calc_Moment** = true(**1**).
  - **Stop_Rotation** $\cdots$ Flag of whether to stop a rotational moment of system
    It is effective only at the time of **Calc_Moment** = true(**1**).

  Notes:
  In **COGNAC**, the rotational moment is calculated based on the coordinates of the imaged atoms, in which the center of gravity of each molecule exists in a unit cell. Note that it differs from the rotation moment obtained from real coordinates.

- **RATTLE**: Setup of RATTLE
  - **Bond** $\cdots$ Flag of whether to apply bond constraint

  - **Bond_Potential_Name** ··· List of **Bond_Potential_Name** where the RATTLE is applied. If the list is empty, RATTLE is applied to all bonds.
  - **Angle** ··· Flag of whether to apply angle constraint
    It is effective, only at the time of **Bond**=true(**1**).
  - **Angle_Potential_Name** ··· List of **Angle_Potential_Name** where the RATTLE is applied. If the list is empty, RATTLE is applied to all angles.
  - **Threshold** ··· Convergence criteria

  - **Use_RATTLE** ··· Flag for appling bond RATTLE at relaxation step. This function is effective to avoid obtaining unrealistic bond length at catenated structure

- **Max_Force** ··· Maximum force acting on each atom.

  A simulation will be aborted if the force acting on an atom becomes larger than this value under the MD simulation. This function is prepared in order not to diverge the simulation and obtain meaningless results.

Solver

Setup for dynamics/minimize

- **Solver_Type** ··· Solver type

  It is selected from **DYNAMICS**/**MINIMIZE**.

- **Dynamics**: Setup for dynamics

  - **Dynamics_Algorithm** ··· Method of dynamics
    It is selected from the following:

    ```
    NVE
    NVT_Nose_Hoover
    NVT_Berendsen
    NVT_Kremer_Grest
    NPH_Andersen
    NPH_Parrinello_Rahman
    NPH_Brown_Clarke
    NPT_Andersen_Nose_Hoover
    NPT_Andersen_Kremer_Grest
    NPT_Parrinello_Rahman_Nose_Hoover
    NPT_Parrinello_Rahman_Kremer_Grest
    NPT_Berendsen
    NPT_Brown_Clarke
    SLLOD_T_Const
    SLLOD_PT_Const
    DPD
    ```

    The meaning of each keyword is explained. Refer to §2.3 for more details.

    * **NVE** ··· Micro canonical ensemble
      It is possible to perform a velocity scaling. Moreover, it is also possible to apply shear flow which is carried out by Lees-Edwards boundary conditions [**?**] in addition to NVE.
    * **NVT_Nose_Hoover** ··· Temperature control by the Nose-Hoover method
    * **NVT_Berendsen** ··· Temperature control by loose-coupling method
    * **NVT_Kremer_Grest** ··· Temperature control by random force (Langevin dynamics)
    * **NPH_Andersen** ··· Pressure control by the Andersen extended Hamiltonian method

* **NPH_Parrinello_Rahman** ··· Anisotropic pressure control by the Parrinello-Rahman extended Hamiltonian method
  It is also possible to vary unit cell lengths independently while unit cell angles is fixed.
* **NPH_Brown_Clarke** ··· Anisotropic pressure control by loose-coupling method
  It is also possible to vary unit cell lengths independently while unit cell angles is fixed.
* **NPT_Andersen_Nose_Hoover** ··· Pressure control by the Andersen extended Hamiltonian method and temperature control by the Nose-Hoover method
  Only isotropic cell deformation is possible.
* **NPT_Andersen_Kremer_Grest** ··· Pressure control by the Andersen extended Hamiltonian method and temperature control by random force
  Only isotropic cell deformation is possible.
* **NPT_Parrinello_Rahman_Nose_Hoover** ··· Pressure control by the Parrinello-Rahman extended Hamiltonian method and temperature control by the Nose-Hoover method
  Anisotropic cell deformation is possible. It is also possible to vary unit cell lengths independently while unit cell angles is fixed.
* **NPT_Parrinello_Rahman_Kremer_Grest** ··· Pressure control by the Parrinello-Rahman extended-Hamiltonian method and temperature control by random force
  It is also possible to vary unit cell lengths independently while unit cell angles is fixed.
* **NPT_Berendsen** ··· Pressure and temperature control by loose-coupling method
  Only isotropic cell deformation. Temperature control follows **NVT_Berendsen** algorithm. Pressure control is carried out with scaling atomic coordinates and unit cell size by μ obtained by eq.2.20.
* **NPT_Brown_Clarke** ··· Pressure and temperature control by loose-coupling method
  Anisotropic cell deformation is possible. It is also possible to vary unit cell lengths independently while unit cell angles is fixed. Temperature control follows **NVT_Berendsen** algorithm. Pressure control follows eq.2.21 which is the extension of the algorithm of Berendsen et al.
* **SLLOD_T_Const** ··· Shear flow by SLLOD+Lees-Edwards boundary conditions, and temperature control by the constraint method
  When adding a shear flow by SLLOD in **COGNAC**, temperature control is always performed by the Nose-Hoover method.
* **SLLOD_PT_Const** ··· Shear flow by SLLOD+Lees-Edwards boundary conditions, and temperature control by the Nose-Hoover method.
  The stress of normal direction is controlled by Parrinello-Rahman method. That is, in the case of shear flow $\dot{\gamma}_{yx}$, in order to control the stress of the direction of z, only c axis of the unit cell will be changed.
  The algorithm for temperature and stress control are changed to Nose-Hoover and Parrinello-Rahman algorithm respectively in Version 8.0.
* **DPD** ··· Dissipative particle dynamics. See section 2.4.

The parameters of each method corresponding to **Dynamics_Algorithm** are listed below.

* **Q** ··· Coupling constant with heat bath used in the case of the temperature control by the Nose-Hoover method
* **tau_T** ··· Coupling constant used in the case of the temperature control by loose-coupling method
* **Friction** ··· Friction constant used in the case of the temperature control by the Kremer-Grest method
* **Cell_Mass** ··· Virtual mass of the unit cell used in the case of the pressure control by the extended Hamiltonian method
  Absolute quantity with the dimension of mass m is taken.
* **Fix_Angle** ··· Flag of whether to fix the unit cell angle used in the case of the anisotropic pressure control method
  Angles are fixed in the case true(**1**).

  - **Fix_Cell** ··· Specification of the axis to fix their length used in the case of the anisotropic pressure control method
    The name of axes, such as **x** (only a axis is fixed), **yz** (bc axis is fixed), **x_iso** (a axis is fixed while the b and c lengths will be changed with keeping the ratio of b/c length), and **iso_xy** (a and b lengths will be changed with keeping the ratio fo a/b length), are specified. It is effective only at the time of **Fix_Angle** = true (**1**)
  - **tau_P** ··· Coupling constant used in the case of the pressure control by loose-coupling method
    In the Berendsen algorithm, the value has the dimension $\tau_p/\beta[\mathrm{ML}^{-1}\mathrm{T}^{-1}]$, which includes isothermal compressibility $\beta[\mathrm{P}(=\mathrm{ML}^{-1}\mathrm{T}^{-2})\mathrm{P}^{-1}]$. (Refer to ref.[**?**]) . In the Brown-Clarke algorithm, the value is equivalent to m of eq.2.21, and has the dimension $[\mathrm{ML}^{-2}\mathrm{T}^{-1}]$. (Refer to ref.[**?**])
  - **lambda** ··· The coefficient for the modified velocity Verlet used in DPD simulation. See eq. 2.37.

- Setup for minimization

  - **Minimize_Algorithm** ··· Method of minimization
    It is selected from the following:

    ```
    Steepest_Descent ... Steepest descent method
    Conjugate_Gradient ... Conjugate gradient method
    Cascade ... Steepest descent and conjugate gradient are performed sequentially.
    ```

  - **Max_Iteration** ··· Maximum number of iteration
  - **Converge_Force** ··· Convergence criteria
    Minimization will be converged, if the all forces acting on atoms become smaller than this value.
  - **Output_Interval_Steps** ··· Interval of data output
  - **Cascade.SD_Max_Interaction** ··· Maximum iteration of steepest descent method in the case of **Cascade** method
  - **Cascade.SD_Converge_Force** ··· Convergence criteria of steepest descent method in the case of **Cascade** method
  - **Strain_Tensor[]** ··· Deformation tensor applied to the initial unit cell
    Sequences of deformation and minimize are repeated by the number of elements of array.

Boundary_Conditions

Setup for boundary conditions

- **a_axis**,**b_axis**,**c_axis** ··· Boundary condition of each axis
  It is selected from **NONE**/**PERIODIC**/ **REFLECTIVE1**/**REFLECTIVE2**

  - **NONE** ··· No boundary conditions
  - **PERIODIC** ··· Periodic boundary conditions
    In **COGNAC**, 2D and 3D periodic boundary conditions are available.
  - **REFLECTIVE1**/**REFLECTIVE2** ··· Staggered reflective boundary condition(§2.8)[**?**]
    This new boundary conditions are developed for modeling interfacial structure. **REFLECTIVE1** only apply the staggered boundary condition on the plane at $r = \mathrm{Cell}_{max}$ and no boundary condition at $r = 0$.
    On the other hand, **REFLECTIVE2** apply the staggered reflective boundary conditions to both planes.

Notes:

  - When the boundary conditions are **PERIODIC** and shear strain and/or shear flow are set, Lees-Edwards boundary conditions are applied automatically.
  - When **NONE** is selected, external wall potentials should be applied on the planes of no boundary codition, since the dynamics of the atom which goes beyond the planes is not supported.

- **Periodic_Bond** ··· Flag for applying boundary conditions for chemical bonds

  The bond lengths, angles and torsions are calculated based on the minimum image distance between atoms in the case of true(**1**).

Calc_Potential_Flags

Flags of whether to calculate potential

- **Bond**, **Angle**, **Torsion**, **Non_Bonding_Inter**, **Non_Bonding_Intra**, **Non_Bonding_1_3**, **Non_Bonding_1_4** **External** and **Electrostatic**

  Flags of whether to calculated each potential term
  Each potential term is calculated at the time of true(**1**). **Non_Bonding_1_3** (non-bonding interaction of 1-3 pair) is effective only when **Angle** and **Non_Bonding_Intra** or **Non_Bonding_Inter** are true(**1**). Namely, **Non_Bonding_1_3** is always calculated at the time of **Angle** is false(**0**). Moreover, **Non_Bonding_1_3** is not calculated at the time of **Non_Bonding_Intra** and **Non_Bonding_Inter** are false(**0**).
  When the electrostatic force is calculated, the calculation of 1-3 electrostatic interaction is turned on and off on the same criteria.
  The effect of **Non_Bonding_1_4** depends on **Torsion** flag in the same manner.

- **Tail_Correction**

  Flag of whether to perform tail correction of non-bonding interaction
  It is effective only in the Lennard-Jones and general Lennard-Jones type non-bonding interaction.

Output_Flags

Flags of whether to output the calculated results

- **Statistics** ··· Output of statistical data, such as energy and pressure

  When each flag is true(**1**), each calculation result, i.e. **Energy**, **Temperature**, **Pressure**, **Stress**, **Volume**, **Density**, **Cell**, **Wall_Press** and **Energy_Flow**, is output respectively.

  Notes:
  This output flags are effective only in a standard output and data file output, and all the items are output to UDF file.

- **Structure** ··· Output of coordinate, velocity and force

  When each flag is true(**1**), **Position**, **Velocity**, and **Force** is output respectively.

- **Averaged_Structure** ··· Output of batch averaged coordinate, velocity and force

  When each flag is true(**1**), averaged value of **Position**, **Velocity** and **Force** during the time steps specified at **Dynamics_Conditions.Time.Output_Interval_Steps** is output respectively.

- **Correlation_Function** ··· Calculation and output of autocorrelation functions. This version support only stress auto correlation.

Density_Output

Setup for the calculation and the output of density distribution from atomic coordinates. Refer to §2.10 for the detail of the method of the calculation of the density distribution.

- **Calc_Grid_Density** ··· Flag of whether to calculate a density distribution
- **Interval_of_Calc_Density** ··· Interval of MD steps to collect data for density distribution
- **Collect_Density_Steps** ··· Number of steps to collect the coordinates data for the calculation of density

  Date sampling is performed from just before the step specified by **Output_Interval** back to **Collect_Density_Steps** with an interval specified by **Interval_of_Calc_Density**.
- **Radius** ··· Effective radius of atom in the case of calculating density distribution
- **Normalize_On** ··· Flag of whether to carry out normalization
- **Num_of_Grid** ··· Number of the lattice which calculates density **(x,y,z)**

Constraint_Conditions

Setting for constraint atom

- **Read_from_Restart** ··· Flag for read constraint conditions from restart file
  Constraint atoms are read from the restart udf specified in **Initial_Structure.Read_Set_of_Molecules.UDF_Name** or argument -R
- **Constraint_Atom**[] ··· Array of atom to constrain
  - **Index** ... Specification of atom to constrain
    - **Mol_Index** ... Molecular index
    - **Atom_Index** ... Atom index
  - **Constraint_Axis** ... Axis to constrain
    - **x,y,z** ··· In the case of **YES**, the dynamics in specified axis is constrained. In the case of **NO**, the dynamics in specified axis is not constrained.
  - **Method** ··· Steady constraint conditions **Steady**/Dynamic constraint conditions **Dynamic**[Keyword]
    Dynamics constraint will constraint the atom with harmonic motion from the initial position.
  - **Steady.Velocity** ··· Velocity of atom to constrain

    The constraint atom will move by the constant velocity during the dynamics simulation. If the **Velocity** is set as **(0.0, 0.0, 0.0)**, the atom is fixed in the initial position.
  - **Dynamic.Amplitude** ··· Amplitude for dynamic constraint
  - **Dynamic.Frequency** ··· Frequency for dynamic constraint

5.2.3 Initial_Structure

Setup for initial structure

Initial_Unit_Cell

Setup for a unit cell

- **Density** ··· Setting density

- **Cell_Size** ··· Cell size

  In the case of a cube and a rectangular cell, **a, b** and **c** of the unit cell correspond to x, y and z axis respectively.

  Notes:
  If **Cell_Size** is set, **Density** will be neglected and the unit cell will be created according to the input of **Cell_Size**. However, there are following exceptions.

  - When the value of **alpha**, **beta** or **gamma** is zero ··· A rectangular cell is created.
  - When the value of any one side of **a**, **b** or **c** is **0.0** ··· The length of cell, which is set as **0.0**, is calculated so that the density may become the specified density. It is effective only at the time of $alpha = beta = gamma = 90.0deg$.
  - When the value of any two sides of **a**, **b** and/or **c** are **0.0** ··· The lengths of cell, which are set to **0.0**, are calculated so that the density may become the specified density. The lengths of two sides are set as an equal length. It is effective only at the time of $alpha = beta = gamma = 90.0deg$.

- **Shear_Strain** ··· Initial shear strain at the time of using Lees-Edwards boundary conditions

Read_Set_of_Molecules

- **UDF_Name** ··· UDF file name to read the data **Set_of_Molecules**
  In the case of blank, **Set_of_Molecules** is read from the input UDF.

- **Record** ··· **Record No.** to read the data **Set_of_Molecules**

  Except for the case of the simulation with considering the reaction, **Set_of_Molecules** is usually output only to the **Initial** data section (it is specified as **Record No.=-1**). When you wish to use the output UDF with reaction, if you set **Record No.** larger value than real number of record of restart UDF, the last record will be used for restart.

Generate_Method

Set up for the method of the preparation of initial structure

- **Method** ··· Method of the preparation of initial structure
  It is selected from the following:

```
Restart ...Coordinates and velocity are read from a specified UDF.
Random ...Amorphous structure is generated.

          When creating multiphase structure, Random is chosen and terms and
          conditions are specified in the option of Random.

Helix ...Helical structures are generated at regular lattice.
Lamella ...Semi-crystalline lamella structure is generated.
Crystal ...Crystal structure is generated from the infomation of lattice
           constant, symmetrical operation, fractional
           coordinate, etc. specified in UDF file.
```

- **Restart**

  - **UDF_Name** ··· UDF file name to read a restart data
    In the case of blank, the data for restart is read from UDF which was specified by **Read_Set_of_Molecules**.

- **Record** ··· **Record No.** to read a restart data
  The data of the record in the restart UDF file is read. When a negative number or a bigger number than the existing number of records is specified, the last record of restart UDF file is read.
- **Restore_Cell** ··· Flag of whether to read the cell size from restart UDF file
- **Restore_Velocity** ··· Flag of whether to read the velocity of atoms from restart UDF file

• **Random**

- **Fix_Angle** ··· Flag of whether to fix angles to the equilibrium angle given by corresponding angle potential
- **Num_Torsion_State** ··· Option which specifies torsion angles
  It is effective in the case that **Fix_Angle** is true (**1**)

```
Num_Torsion_State = 0 ... Generate at random
Num_Torsion_State = 1 ... Generate only trans conformation
Num_Torsion_State = 2 ... Generate trans-cis conformation
Num_Torsion_State = 3 ... Generate trans-gauche conformation
```

- **Delta_E** ··· Energy difference between *trans−cis* or *trans−gauche* in the case of **Num_Torsion_State** =**2** or **3** respectively
  The energy difference of *cis* or *gauche* states from the *trans* state is specified.
- **Build_Temp** ··· Temperature which generates torsion angles in **Num_Torsion_State** = **2** or **3**
  The probability of each state is calculated from the Boltzmann factor using **Delta_E** and **Build_Temp**.
- **Density_Bias**[]: Generation of the initial structure by the density biased Monte Carlo method based on atom type
  - **Atom_Name** ··· Atom type name to apply bias
  - **UDF_Name** ··· **SUSHI** UDF file name with a volume fraction
  - **Component_Index** ··· Segment index of the volume fraction in **SUSHI** (index of **SUSHIOutput.phi.value[].comp[]**), **Muffin_phaseseparation** (index of **field.scalar_field[].value[].comp[]**) or **COGNAC** (index of **Grid_Density.phi.value[].comp[]**) to calculate potential
  - **Max_Retry** ··· Number of steps of the maximum trial of Monte Carlo step

- **Node_Density_Bias**[]: Generation of the initial structure by the node density biased Monte Carlo method based on the spatial distribution of segments in a chain.
  - **Molecular_Name** ··· Molecular name to apply bias
  - **UDF_Name** ··· **SUSHI** UDF file name with segment volume fractions
  - **Start_Index** ··· Index of segment in **SUSHI** UDF file corresponding to the first atom in the molecule
  - **End_Index** ··· Index of segment in **SUSHI** UDF file corresponding to the last atom in the molecule
  - **Scale_Param** ··· Ratio of MD atom and SCF segment

    For example, if 1 atom of MD corresponds to 1 segment of SCF such as the case of bead-spring model, the **Scale_Param** is set as **1.0**. When more microscopic MD model is used, the **Scale_Param** must be set as a larger value.

  - **Max_Retry** ··· Number of steps of the maximum trial of Monte Carlo step

- **Meso_Ratio** ··· Probability of meso diad for chiral atoms specified in **Main_Chain**
  It will be ignored if chirality is specified in **Set_of_Molecules**.

- **Fix_End_Position**[] : Specify the position of begining atom (Atom Index = **0**) of molecules.

  - * **Mol_Index** ··· Index of molecules
  - * **Position** ··· Position of begining atom(x,y,z)

- **Helix**
  - **Lattice_Type**: Setup for lattice information
    - * **Type** ··· Setup of lattice type, e.g. simple cubic lattice, body-centered cubic lattice
      It is selected from the following:

      ```
      SC ... Simple cubic lattice
      FCC ... Face-centered cubic lattice
      BCC ... Body-centered cubic lattice
      Manual ... Although it is like the body-centered cubic lattice,
                 the shift value can be specified arbitrarily.
      ```

    - * **Manual.X_Shift** ··· Amount of shifts in the direction of x-axis of the atoms that are in the odd number layer of y-axis
      The value is normalized by the interval of lattice in the direction of z-axis.
    - * **Manual.Z_Shift** ··· Amount of shifts in the direction of z-axis of the atoms that are in the odd number layer of y-axis
      The value is normalized by the interval of lattice in the direction of z-axis.
      In the case of **X_shift** = **Z_shift** = 0.5, the structure becomes the same as BCC.
  - **Num_X_Cell** ··· Number of unit of lattice in the direction of x(a) axis
  - **Num_Y_Cell** ··· Number of unit of lattice in the direction of y(b) axis

    Notes:

    1. When **Lattice_Type.Type** is **Manual**, **Num_X_Cell** and **Num_Y_Cell** mean numbers of molecule in x and y direction respectively, and they may not agree with the number of lattice.
    2. The size of lattice is calculated by the size of unit cell and **Num_X/Y_Cell**. Therefore, non cubic lattice can be created, even if it specifies as **Type='SC'** for example.
    3. The number of unit of lattice in the direction of z axis is automatically calculated by **Num_X/Y_Cell** and the number of total molecules.

  - **Fix_Angle** ··· Flag of whether to fix angles to the equilibrium angle given by corresponding angle potential
  - **Torsion**[]··· Torsion angle of helical structure. $trans = 0$
    In the case of helix defined by multiple torsion angles in sequence, the array has multiple torsion angles in order from the first torsion.
  - **Rotate1** ··· Rotation angle of the molecule around z-axis of the even number layer of y-axis
  - **Rotate2** ··· Rotation angle of the molecule around z-axis of the odd number layer of y-axis
  - **Orient_Vector**[]··· Pair of atom index defining the vector to order in the direction of z-axis
    For example, in the case of the polyethylene united atom defined as C0-C1-C2-C3-C4- ..., if **Orient_Vector[0]** and **Orient_Vector[1]** are specified as 0 and 2 respectively, and the modeling of helix of all trans conformation is carried out, the helix which oriented C0-C2-C4 in the direction of z-axis is generated. A result is the same in the case that it is specified as 0 and 4. When this specification does not exist, the vector between 2 atoms of maximum index and minimum index will be oriented to z-axis.
  - **Meso_Ratio** ··· Probability of meso diad for chiral atoms specified in **Main_Chain**
    It will be ignored if chirality is specified in **Set_of_Molecules**.
  - **Inverse_Molecules** ··· Option for the inversion of molecules.
    - * **NONE** ··· No inversion
    - * **Neighbour_by_Neighbour** ··· Inverse molecules neighbout by neighbour

  - **Line_by_Line** ··· Inverse molecules line by line in the direction of b-axis
  - **Layer_by_Layer** ··· Inverse molecules layer by layer in the direction of c-axis

- **Lamella**
  - **Lamella_Length** ··· Length of a crystalline phase of lamella (in the direction of z-axis)
  - **Fix_Angle** ··· Flag of whether to fix angles to the equilibrium angle given by corresponding angle potential
  - **Meso_Ratio** ··· Probability of meso diad for chiral atoms specified in **Main_Chain**
    It will be ignored if chirality is specified in **Set_of_Molecules**.
  - **Random_Param.Density** ··· Density of an amorphous phase
  - **Random_Param.Num_Torsion_State** ··· Option which specifies a torsion angle
    It is effective in the case that **Fix_Angle** is true (**1**)
  - **Random_Param.Delta_E** ··· Energy difference between $trans - cis$ or $trans - gauche$ in the case of **Num_Torsion_State** =**2** or **3** respectively
    The energy difference of *cis* or *gauche* states from the *trans* state is specified.
  - **Random_Param.Build_Temp** ··· Temperature which generates torsion angles in **Num_Torsion_State** = **2** or **3**
    The probability of each state is calculated from the Boltzmann factor using **Delta_E** and **Build_Temp**.
  - **Random_Param.Scale_Param** ··· Ratio of MD atom/SCF segment
  - **Helix_Param.Density** ··· Density of a crystalline phase
  - **Helix_Param.Torsion**[] ··· Torsion angle of helical structure. $trans = 0$
    In the case of helix defined by multiple torsion angles in sequence, the array has multiple torsion angles in order from the first torsion.
  - **Helix_Param.Helix_Length** ··· Estimated length of helix per 1 atom
    The rough estimation is set up in order to calculate the chain density of the crystalline phase.

- **Crystal**
  - **UDF_Name** ··· UDF file name with crystal structure data
  - **Num_X_Cell** ··· Number of lattice unit in the direction of a-axis
  - **Num_Y_Cell** ··· Number of lattice unit in the direction of b-axis
  - **Num_Z_Cell** ··· Number of lattice unit in the direction of c-axis
  - **Unit_Length** ··· Ratio of the cell size specified by crystal structure data and the unit of the length actually used in **COGNAC**
    For example, when crystal structure data is described in a real unit such as angstrom(Å), The **Unit_Length** is set as a reduced length in the unit (Å).

    Since the unit lattice data is read from UDF file in **Crystal** option, and multiplied by the number of **Num_X/Y/Z_Cell**, specified unit cell size in input is overwritten.

Relaxation

Setup for the relaxation of initial structure

- **Relaxation** ··· Flag of whether to perform energy relaxation of initial structure
  It is started after initial structure is read or generated.
- **Method** ··· Method of initial structure relaxation
  It is selected from **DYNAMICS/MINIMIZE**.
- **Max_Relax_Force** ··· Initial value of maximum force acting on atom
  This value should be set not to diverge a following MD simulation, after the relaxation is finished.
- **Max_Relax_Steps** ··· Maximum steps of relaxation

5.2.4 Molecular_Attributes

The information of type of atom and interaction site, and bond, angle, and torsion potential for simulations is defined.

The array type data is defined according to the number of type/potential used for simulations.

Atom_Type[]

- **Name** ··· Name of atom type
  Arbitrary name can be defined, unless duplicated.
- **Mass** ··· Mass of the atom type

Bond_Potential[]

- **Name** ··· Name of bond potential
  Arbitrary name can be defined, unless duplicated.
- **Potential_Type** ··· Type of the bond potential
  It is selected from the following:

  ```
  Harmonic
  FENE_LJ
  Gauss
  Morse
  Bond_Polynomial
  DPD
  Table_Bond
  User_Bond
  ```

  Refer to §2.6.1 for the detail of each type.

- **R0** ··· Equilibrium bond length

  Required parameters for each **Potential_Type** are explained below.

  - **Harmonic**
    - **K** ··· Spring constant
  - **FENE_LJ**
    - **R_max** ··· Maximum length of FENE term
    - **K** ··· Spring constant of FENE term
    - **sigma** ··· Length parameter of LJ term
    - **epsilon** ··· Strength of interaction in LJ term
  - **Gauss**
    - **Temperature** ··· Temperature to calculate the potential
      Different value from simulation temperature can be set.
  - **Morse**
    - **A** ··· Parameter A used by eq.2.47 or 2.48
    - **B** ··· Parameter B used by eq.2.47 or 2.48

  - **Zero_at_Infinity** ··· In the case **NO**, eq.2.47 is used. In the case **YES**, eq.2.48 is used.

- **Bond_Polynomial**
  - **N** ··· Order of polynomial + 1
  - **p[]** ··· Array of coefficients p[0] – p[N-1]

- **DPD**
  - **C** ··· Spring constant

- **Table_Bond**
  - **UDF_Name** ··· Table UDF file name
  - **Use_Fast** ··· Fast table will be used in the case that **YES** is selected
    If **YES**, the parameter **Method** specify the method of obtaning the ebergy value,
    - **Interpolation** ··· Energy is calulated by the linear interpolation,
    - **Nearest_Value** ··· Energy is obtained from the nearest table value.

    If **NO**, **Order** specify the order of interpolation to calculate energy and force from a table

- **User_Bond**
  - **Index** ··· Index of the class **UserBond** to be used
  - **Parameter[].Name** ··· Name of parameter
    Arbitrary name can be defined, unless duplicated.
  - **Parameter[].Value** ··· Value of parameter

Angle_Potential[]

- **Name** ··· Name of angle potential
  Arbitrary name can be defined, unless duplicated.
- **Potential_Type** ··· Type of the angle potential
  It is selected from the following:

```
Theta
Theta2
Cosine
Theta_Polynomial
Table_Angle
User_Angle
```

  Refer to §2.6.2 for the detail of each type.

- **theta0** ··· Equilibrium angle (the value in degree)

  Required parameters according to the type keyword of **Angle_Potential** are explained below.

  - **Theta/Theta2/Cosine**
    - **K** ··· Spring constant (The unit will be ϵ/rad^2 for **Theta** type potential, while ϵ for others.)

  - **Theta_Polynomial**
    - **N** ··· Order of polynomial + 1

    * **p[]** ··· Array of coefficients p[0] – p[N-1]

  – **Table_Angle**
    * **UDF_Name** ··· Table UDF file name
    * **Use_Fast** ··· Fast table will be used in the case that **YES** is selected
      If **YES**, the parameter **Method** specify the method of obtaning the ebergy value,
      · **Interpolation** ··· Energy is calulated by the linear interpolation,
      · **Nearest_Value** ··· Energy is obtained from the nearest table value.
      If **NO**, **Order** specify the order of interpolation to calculate energy and force from a table

  – **User_Angle**
    * **Index** ··· Index of the class **UserAngle** to be used
    * **Parameter[].Name** ··· Name of parameter
      Arbitrary name can be defined, unless duplicated.
    * **Parameter[].Value** ··· Value of parameter

Torsion_Potential[]

- **Name** ··· Name of torsion angle potential
  Arbitrary name can be defined, unless duplicated.
- **Potential_Type** ··· Type of the torsion angle potential
  It is selected from the following:

```
Cosine_Polynomial
Amber
Dreiding
Table_Torsion
User_Torsion
```

  Refer to §2.6.3 for the details of each type.

  Required parameters according to the type keyword of **Torsion_Potential**, are explained below.

  – **Cosine_Polynomial**
    * **K** ··· Coefficient to all the term of polynomial
    * **N** ··· Order of polynomial + 1
    * **p[]** ··· Array of coefficients p[0] – p[N-1]

  – **Amber**
    * **PK** ··· One-half of the barrier magnitude
    * **IDIVF** ··· Total number of torsions about a single bond that the potential applies to
    * **PN** ··· Periodicity. For example, **PN**=2 means that the torsion potential has two-fold periodicity
    * **PHASE** ··· Phase shift. For example, the torsion potential has a maximum at 0 degree, if **PHASE**=0
    * **trans_is_0** ··· If **trans_is_0** is set 0 (false), the torsion angle is defined 0 degree at cis conformation following to the original Amber potential.

  – **Dreiding**

    * **V** ··· Barrier magnitude
    * **phi0** ··· Equilibrium angle. For example, the torsion potential has a minimum at 0 degree, if **phi0**=0
    * **n** ··· Periodicity. For example, **n**=2 means that the torsion potential has two-fold periodicity
    * **trans_is_0** ··· If **trans_is_0** is set 0 (false), the torsion angle is defined 0 degree at cis conformation following to the original Dreiding potential.

  – **Table_Torsion**
    * **UDF_Name** ··· Table UDF file name
    * **Use_Fast** ··· Fast table will be used in the case that **YES** is selected
      If **YES**, the parameter **Method** specify the method of obtaning the ebergy value,
      · **Interpolation** ··· Energy is calulated by the linear interpolation,
      · **Nearest_Value** ··· Energy is obtained from the nearest table value.
      If **NO**, **Order** specify the order of interpolation to calculate energy and force from a table

  – **User_Torsion** a
    * **Index** ··· Index of the class **UserTorsion** to be used
    * **Parameter[].Name** ··· Name of parameter
      Arbitrary name can be defined, unless duplicated.
    * **Parameter[].Value** ··· Value of parameter

Interaction_Site_Type[]

- **Name** ··· Name of interaction site
  Arbitrary name can be defined, unless duplicated.
- **Num_of_Atoms** ··· Number of atom(s) to define the interaction site
- **Range** ··· Radius for making neighbor list

  In the case of the distance between a pair of interaction site is smaller than the sum of **Range** of the interaction sites, the pair is registered in a neighbor list.
  Usually, it is set as $\mathbf{Range_{site1}} + \mathbf{Range_{site2}} = cutoff + \alpha$. *alpha* is a buffer length between the maximum distance of neighbor list and cutoff distance of non bonding interaction. If the *alpha* becomes larger, the frequency of update of neighbor list will decrease, but the size of pair list for calculating non-bonding interaction will increase. α should be determined to maximize the efficiency of the calculation of non-bonding interaction.
  If the **Range** is set as 0.0, the site never be registered in neighbor list and used only for external interaction or electrostatic interaction.

Electrostatic_Site_Type[]

- **Name** ··· Name of electrostatic site type
  Arbitrary name can be defined, unless duplicated.
- **Type_Name** ··· Type name of electrostatic site type. It is selected from the following:

  ```
  POINT_CHARGE
  CENTER_DIPOLE
  END_DIPOLE
  ```

- **Polarizability** ··· Polarizability of electrostatic site type. This parameter is used for field electrostatic method.

5.2.5 Interactions

The information of non-bonding, external and electrostatic interactions used for simulations is defined.
The array type data is defined according to the number of potential used for simulations.

Pair_Interaction[]

- **Name** ··· Name of pair interaction
  Arbitrary name can be defined, unless duplicated.
- **Potential_Type** ··· Type of the pair interaction
  It is selected from the following:

  ```
  Lennard_Jones
  Lennard_Jones_EV
  General_Lennard_Jones
  Gay_Berne
  GB_LJ
  DPD
  Morse
  Buckingham
  Table_Pair_Potential
  User_Pair_Interaction
  ```

  Refer to §2.6.4 for the details of each type, and the meaning of the parameters.
- **Site1_Name/Site2_Name** ··· A pair of name of interaction site to apply the pair interaction
- **Cutoff** ··· Cutoff distance
- **Scale_1_4_Pair** ··· Scale factor for 1-4 pair interaction. Effective when including torsion potential and **Simulation_Conditions.Calc_Potential_Flags.Non_Bonding_1_4** is true.

  Required parameters according to **Potential_Type** of **Pair_Interaction** are explained below.

  - **Lennard_Jones**
    - **sigma** ··· Diameter of LJ sphere Refer to σ in eq.2.58.
    - **epsilon** ··· Strength of interaction Refer to ϵ in eq.2.58.
  - **Lennard_Jones_EV**
    - **sigma** ··· Diameter of LJ sphere Refer to σ in eq.2.59.
    - **epsilon** ··· Strength of interaction Refer to ϵ in eq.2.59.
    - **R_EV** ··· Diameter of excluded volume Refer to R_{EV} in eq.2.59
  - **General_Lennard_Jones**
    - **sigma** ··· Diameter of LJ sphere Refer to σ in eq.2.60.
    - **epsilon** ··· Strength of interaction Refer to ϵ in eq.2.60.
    - **A** ··· Coefficient of of repulsive term Refer to A in eq.2.60.
    - **B** ··· Coefficient of of disspersive term Refer to B in eq.2.60.
    - **m** ··· Power of repulsive term Refer to m in eq.2.60.
    - **n** ··· Power of disspersive term Refer to n in eq.2.60.
  - **Gay_Berne**
    - **sigma** ··· Size parameter σ_0. Refer to eqs.2.61 and 2.62.

* **epsilon** ··· Energy parameter ϵ. Refer to eq.2.66.
* **l1**, **l2** ··· l_i and l_j in eqs.2.64 and 2.65
* **d1**,**d2** ··· d_i and d_j in eqs. 2.63, 2.64 and 2.65
* **mu,nu** ··· μ and ν in eq.2.66
* **alpha2,k2** ··· 墜家調墜翁墜俺調 eq.2.68 α' and eq.2.69 k'

– **GB_LJ**

* **sigma** ··· Size parameter σ_0. Refer to eqs.2.70 and 2.71.
* **epsilon** ··· Energy parameter ϵ. Refer to eq.2.72.
* **l2** ··· l_j in eq.2.75.
* **d1**,**d2** ··· d_i and d_j in eqs.2.74 and 2.75.
* **k2** ··· k' in eq.2.75.
* **mu** ··· μ in eqs.2.72 and 2.75.

– **DPD**

* **a** ··· Interaction parameter. Refer to eq.2.76
* **gamma** ··· Friction constant. Refer to eq.2.34

– **Morse**

* **A** ··· A in eq.2.77
* **B** ··· B in eq.2.77
* **r0** ··· r_0 in eq.2.77

–
* **A** ··· A in eq.2.78
* **B** ··· B in eq.2.78
* **C** ··· C in eq.2.78

– **Table_Pair_Potential**

* **UDF_Name** ··· Table UDF file name
* **Use_Fast** ··· Fast table will be used in the case that **YES** is selected
  If **YES**, the parameter **Method** specify the method of obtaning the ebergy value,
  · **Interpolation** ··· Energy is calulated by the linear interpolation,
  · **Nearest_Value** ··· Energy is obtained from the nearest table value.
  If **NO**, **Order** specify the order of interpolation to calculate energy and force from a table

– **User_Pair_Interaction**

* **Index** ··· Index of the class **UserPairInteraction** to be used
* **Parameter[].Name** ··· Name of parameter
  Arbitrary name can be defined, unless duplicated.
* **Parameter[].Value** ··· Value of parameter

External_Interaction[]

- **Name** ··· Name of external interaction
  Arbitrary name can be defined, unless duplicated.
- **Potential_Type** ··· Type of the external interaction
  It is selected from the following:

```
LJ_Wall
LJ_Atomic_Wall
Static_Field
Density_Field
Velocity_Field
Total_Density_Constrain External_Angle
External_Torsion
```

Refer to §2.6.5 for the details of each type.

- **Site_Name** ··· Name of the interaction site to apply the external interaction

Required parameters according to the **Potential_Type** of **External_Interaction** are explained below.

- **LJ_Wall**: Potential of flat wall(s)
  - **Cutoff** ··· Cutoff distance
    When the **Cutoff** is set **0**, cutoff is not applied, and the interaction between all atoms and wall will be calculated.
  - **sigma** ··· Diameter of LJ sphere
  - **epsilon** ··· Strength if interaction
  - **Density** ··· Surface density specified by the density on the surface
  - **Direction** ··· Direction of the wall [Keyword]
    If **'z'** is specified, walls will be set on xy plane at **z**=0 and **z**=max. If **'zl'** or **'zu'** is specified, the wall potential is set on only lower plane or upper plane of the unit cell respectively. **'x(l/u)'** and **'y(l/u)'** can be used similarly.

- **LJ_Atomic_Wall**: Potential of structured wall(s)
  - **Cutoff** ··· Cutoff distance
  - **sigma** ··· Diameter of LJ sphere
  - **epsilon** ··· Strength of interaction
  - **Density** ··· Surface density specified by the density on the surface
  - **Position** ··· Position of wall. In the case of **LJ_Atomic_Wall**, walls can be set on xy plane at **z**=0 and **z**=max. [Keyword]
    ```
    Both_Side
    Lower_Side
    Upper_Side
    ```
  - **Shear** ··· Shear rate $\dot{\gamma}_{xz}$
    In the option **LJ_Atomic_Shear**, only shear $\dot{\gamma}_{xz}$ is available.

- **Static_Field**
  - **Field** ··· Field value
    It is specified by 3D vector value.

- **Density_Field**: Density field
  - **UDF_name** ··· UDF file name of **SUSHI**, **Muffin_phaseseparation** which have volume fraction data , or **COGNAC** which has a **Grid_Density** data. If no **UDF_name** is specified, the calculated denisty field under the condition of **Density_Output** is used.
  - **Component_Index** ··· Segment index of the volume fraction in **SUSHI** (index of **SUSHIOutput.phi.value[].comp[]**), **Muffin_phaseseparation** (index of **field.scalar_field[].value[].comp**) or **COGNAC** (index of **Grid_Density.phi.value[].comp[]**) to calculate potential

* **Potential_Type** ··· Potential type [Keyword]
  Refer to section2.6.5 for the detail of the potential.

```
Density_Biased_Potential
Lennard_Jones
Reciprocal_Power
```

  · **Density_Biased_Potential** : Paramter for density biased potential
    chi ··· χ parameter
  · **Lennard_Jones** 墜家調墜翁墜俺調 Parameter for Lennard Jones type potential
    sigma ··· Lennard Jones diameter
    epsilon ··· Lennard Joned energy
  · **Reciprocal_Power** 墜家調墜翁墜俺調 Parameter for reciprocal power type potential
    Order ··· Order of the potential
    Coefficient ··· Coefficient

– **Total_Density_Constrain**: Potential for total density constraint

* **coeff** ··· Coefficient of potential

– **External_Angle**: Cosine type angle potential acting on three atoms site which exists in the specified region in a space

* **theta0** ··· Equilibrium angle
* **K** ··· Spring constant
* **Min_Position** ··· Minimum position to specify the region **(x, y, z)**
* **Max_Position** ··· Maximum position to specify the region **(x, y, z)**

– **External_Torsion**: Cosine polynomial type torsion potential acting on four atoms site which exists in the specified region in a space

* **phi0** ··· Shift angle of torsion angle potential
* **K** ··· Coefficient to all the term of polynomial
* **N** ··· Order of polynomial + 1
* **p[]** ··· Array of coefficients p[0] – p[N-1]
* **Min_Position** ··· Minimum position to specify the region **(x, y, z)**
* **Max_Position** ··· Maximum position to specify the region **(x, y, z)**

– **Tethered_Force**

* **K** ··· Spring constant

– **User_External_Field**

* **Index** ··· Index of the class **UserExternalField** to be used
* **Parameter[].Name** ··· Name of parameter
  Arbitrary name can be defined, unless duplicated.
* **Parameter[].Value** ··· Value of parameter

Electrostatic_Interaction[]

- **Name** ··· Name of electrostatic interaction
  Arbitrary name can be defined, unless duplicated.
- **Algorithm** ··· Method to calculate the electrostatic interaction
  It is selected from the following:

```
Cutoff_Coulomb
Cutoff_Coulomb_Debye
Reaction_Field
Ewald
Field_Electrostatic
PPPM
```

Refer to §2.6.6 for the details of each type.

- **Scale_1_4_Pair** ··· Scale factor for 1-4 electrostatic interaction. Effectie for **Cutoff_Coulomb** and **Reaction_Field** of point charge. Also Effective when including torsion potential and **Simulation_Conditions.C** is true.

  Required parameters according to **Potential_Type** of **Electrostatic_Interaction** are explained below.

  – **Cutoff_Coulomb**
    * **Dielectric_Constant** ··· Dielectric constant of the system
    * **cutoff** ··· Cutoff distance

  – **Cutoff_Coulomb_Debye**
    * **Dielectric_Constant** ··· Dielectric constant of the system
    * **cutoff** ··· Cutoff distance
    * **kappa** ··· Inverse of Debye length

  – **Reaction_Field**
    * **Dielectric_Constant** ··· Dielectric constant of the surrounded media. If the value is less than 1.0, dielectric constant of the surrounded media is considered to ∞.
    * **cutoff** ··· Cutoff distance

  – **Ewald**
    * **Dielectric_Constant** ··· Dielectric constant of the surrounded media. If the value is less than 1.0, dielectric constant of the surrounded media is considered to ∞.
    * **R_cutoff** ··· Cutoff distance of real space term
    * **Ewald_Parameters** ··· Selection for the setting method of parameters [Keyword]

      ```
      Auto
      Manual
      ```

      · **Auto** ··· Automatic parameter setting
      · **Manual** ··· Manual parameter setting
      alpha ··· Parameter showing the spread of an electric charge distribution

      Note: The parameter **al** in the version before 6.0 is described as

      al = **alpha** * (unit cell length).

      Since **COGNAC** begin to support rectangular cell in Ewald summation from version 6.1, **alpha** instead of **al** is taken as a input parameter

      K_cutoff ··· Parameter showing the cutoff for reciplocal space. **nh,nk** and **nl** correspond the three direction of the space.

  – **Field_Electrostatic**[**?**]

    * **Dielectric_Constant** ··· Dielectric constant of the system
    * **gamma** ··· Scaling parameter of charge
    * **range** ··· Range of Charge smearing
    * **zeta** ··· Relaxation parameter in solving the Poisson equation. A recommended value is **0.15**.
    * **error** ··· Accepted error in solving the Poisson equation. A recommended value is **0.03**.
    * **max_iteration** ··· Maximum iteraction in solving the Poisson equation.
  - **PPPM**[**?**, **?**]
    * **Dielectric_Constant** ··· Dielectric constant of the system
    * **R_cutoff** ··· Cutoff distance of real space term
    * **PPPM_Parameters** ··· Selection for the setting method of parameters [Keyword]

      ```
      Auto
      Manual
      ```

      · **Auto** ··· Automatic parameter setting
      · **Manual** ··· Manual parameter setting

      alpha ··· Parameter showing the spread of an electric charge distribution. Same as Ewald.

      Number_of_Grid ··· Number of mesh for solving Poisson's equation. **nx,ny** and **nx** correspond the number of division in eash axis.
    * **error** ··· Accepted error in solving the Poisson equation. A recommended value is **0.03**.
    * **max_iteration** ··· Maximum iteraction in solving the Poisson equation.

5.2.6 React_Conditions

Setup for chemical reaction (creation and scission of bonds).

- **React_Flag** ··· Flag of whether to activate chemical reaction
  It is selected from **"ON"/"OFF"**.
- **Atom_Exchange** ··· Setup for Atom type exchange
  - **Exchange_Type_Array**[] ··· Array of the setup for Atom type exchange
    * **Interaval_of_Reaction** ··· Interval of time step to judge atom type exchange
    * **Probability** ··· Probability of atom type exchange. The value of $0 - 1.0$ is accepted.
    * **Target_Atom_Type** ··· Target atom type name to exchange
    * **Product_Atom_Type** ··· Atom type name after exchange
    * **Product_Atom_Name** ··· Atom name after exchange. When it is blank, atom name won't be changed.
    * **Product_Interaction_Site_Type** ··· Interaction site type name after exchange
    * **Exchange_Type** ··· Setup for the rule of atom type exchange
      · **Type_Keyword** ··· Type of atom type exchange
      Only **'Region'** is available.
      · **Region** ··· Atoms in a specified region will be exchanged. The position of image atoms in a unit cell can be considered for the exchange.

      Min_Position ··· Minimum position of the region. **x,y** and **z** coordinates are specified.
      Max_Position ··· Maximum position of the region. **x,y** and **z** coordinates are specified.

      If both **Min_Position** and **Max_Position** are set **0**, all coordinate of the axis is specified.

- **Bond_Creation** ··· Setup for bond creation
  - **Reactive_Atom**[] ··· Setup for reactive atoms
    The array data has a number of elements corresponding to the number of the reactive atom types.
    - **Atom_Type_Name** ··· Name of reactive atom type
      A name of atom type defined in **Molecular_Attribute** must be set.
    - **Max_bond_Num** ··· Number of maximum bonds of the reactive atom.
  - **Creation_Type_Array**[] ··· Array of setup for bond creation.
    - **Interval_of_Reaction** ··· Interval of time step to judge bond creation
    - **Probability** ··· Probability of bond creation. The value of $0 - 1.0$ is accepted.
    - **In_Chain** ··· The flag of whether to allow the reaction in a molecule
      It is selected from **"ON"/"OFF"**
    - **Creation_Type** ··· Setup for the rule of bond creation
      - **Type_Keyword** ··· Selection of the rule of bond creation
        'Simple_Creation' and **'Polymerization'** are avairable
      - **Simple_Creation** ··· Conditions of **Simple_Creation**
        Threshold_Distance ··· Threshold distance to judge bond creation
        Potential_Name ··· Name of bond potential for the created bond. A name of bond potential defined in **Molecular_Attribute** must be set
        Atom_Type_Sequence ··· A pair of atoms to react. They are specified in **atom1**/**atom2**. If **atom2** is blank, the reaction between **atom1** and **atom1** is set.
      - **Polymerization** ··· Conditions of **Polymerization**
        Threshold_Distance ··· Threshold distance to judge bond creation
        Potential_Name ··· Name of bond potential for the created bond. A name of bond potential defined in **Molecular_Attribute** must be set **Atom_Type_Sequence** ··· A set of atom type for polymerization
        Reactive_Atom··· Atom type of active site
        Monomer··· Atom type that react with **Reactive_Atom**
        New_Reactive_Atom··· Newly generated active site from **Mmonomer**
        New_Inactive_Atom··· Newly generated inactive site from **Reactive_Atom**

        Atom_Name_Sequence ··· Atom name after polymerization. If it is blank, atom name won't be changed.
        New_Reactive_Atom··· Atom name of newly generated active site from **Monomer**.
        New_Inactive_Atom··· Atom name of newly generated inactive site from **Reactive_Atom**.

        Maximum_Chain_Length··· Maximum chain length. In the case zero, no restriction is applied.
  - **Potential_Assignment** ··· Setting for newly defined angle and torsion at bond creation
    - **Angle**[] ··· Array of setting for newly defined angle potential
      - **Potential_Name** ··· Name of angle potential of created angle
        A name of angle potential defined in **Molecular_Attribute** must be set.
      - **Atom_Type_Sequence** ··· A set of atom types to define new angle
        They are specified in **atom1**/**atom2**/**atom3**.
      - **Bond_Num_of_atom2** ··· Number of bond from **atom2**
        In the case that number of bonds including new bond equals to this value, the angle potential with name **Potential_Name** is assigned to the angle.
    - **Torsion**[] ··· Array of setting for newly defined torsion potential

· **Potential_Name** ··· Name of angle potential of created torsion
A name of torsion potential defined in **Molecular_Attribute** must be set.
· **Atom_Type_Sequence** ··· A set of atoms to define new torsion
They are specified in **atom1**/**atom2**/**atom3**/**atom4**.
· **Bond_Num_of_atom2** ··· Number of bond from **atom2**
· **Bond_Num_of_atom3** ··· Number of bond from **atom3**
In the case that numbers of bonds of **atom2** and **atom3** including new bond equal to these values, the torsion potential with name **Potential_Name** is assigned to the torsion.

- **Scission** ··· Setup for bond scission
  - **Bond_Scission**[] ··· Setup for breakable bond
    The array has a number of elements corresponding to the number of the breakable bond potentials.
    * **Bond_Potential_Name** ··· Name of breakable bond potential
      A bond potential name defined in **Molecular_Attribute** must be set.
    * **Scission_Type** ··· Criterial of bond scission. **Length** and **Region** are supported.
    * **Length** ··· Parameter when **Scission_Type** is **"Length"**
      · **Scission_Length** ··· Threshold distance to judge the bond scission
    * **Region** ··· Parameters when **Scission_Type** is **"Region"**
      · **Min_Position** ··· Minumum position **x,y,z** of rectanguler region.
      · **Max_Position** ··· Maximum position **x,y,z** is rectangular region.

5.2.7 Set_of_Molecules

Set_of_Molecules has the following hierarchical structures.

```
Set_of_Molecules---molecule[]---atom[]
                               |-bond[]
                               |-angle[]
                               |-torsion[]
                               |-interaction_Site[]
                               |-electrostatic_Site[]
```

The parameters in each array are explained below.

molecule[]

- **Mol_Name** ··· Name of molecule
  Usually the same name is set to the same type of molecule.

atom[]

- **Atom_ID** ··· ID of atom
  Arbitrary integer can be set unless duplicate.
- **Atom_Name** ··· Name of atom
  Arbitrary name can be set.
- **Atom_Type_Name** ··· Name of atom type
  A name of atom type defined in **Molecular_Attributes** must be set to each atom.
- **Chirality** ··· Setting of chirality
  It is selected from **1** (R or S) / **0** (random) / **-1** (S or R)

  Notes:
  The exact stereo configuration of R/S is not considered. In the case of the pair of **1** – **1** or **-1** – **-1**, it is meso and in the case of the pair of **1** – **-1**, it is racemic.

- **Main_Chain** ··· Flag for whether to define the atom as a part of main chain
  When the main chain needs to be defined, for example, to generate helix, atom in the main chain is set as true(**1**).

- **Attributes**[] ··· Atributes of atoms. The objects contains **Name** and **Value**, and arbitral attributes can be assigned. For examples, name of monomer unit in which the atom belong are given and used for drawing and zooming. **COGNAC** itself does not use the object.

bond[]

- **Potential_Name** ··· Name of bond potential
  A name of bond potential defined in **Molecular_Attributes** must be set to each bond.

- **atom1**/**atom2** ··· Indexes of atom to define the bond
  Since the target atoms should exist in the same molecule, only indexes of array **atom**[] are specified.

- **Order** ··· Bond order. This variable is used for drawing and assigning force field parameter. **COGNAC** itself does not use the object.

angle[]

- **Potential_Name** ··· Name of angle potential
  A name of angle potential defined in **Molecular_Attributes** must be set to each angle.

- **atom1**/**atom2**/**atom3** ··· Indexes of atom to define the angle
  Since the target atoms should exist in the same molecule, only indexes of array **atom**[] are specified.

torsion[]

- **Potential_Name** ··· Name of torsion potential
  A name of torsion potential defined in **Molecular_Attributes** must be set to each torsion.

- **atom1**/**atom2**/**atom3**/**atom4** ··· Indexes of atom to define the torsion. Since the target atom should exist in the same molecule, only indexes of array **atom**[] are specified.

interaction_Site[]

- **Type_Name** ··· Name of interaction site
  A name of interaction site defined in **Molecular_Attributes** must be set to each interaction site.

- **atom**[] ··· Array of index of atom to define the interaction site
  Since the target atom should exist in the same molecule, only index(es) of array **atom**[] are specified.

electrostatic_Site[]

- **Type_Name** ··· Name of the type of site to apply electrostatic interactions
  If it is selected from the following, the type is defined from the keyword.

  ```
  POINT_CHARGE
  CENTER_DIPOLE
  END_DIPOLE
  ```

  Also, by specifying the **Name** defined in **Electrostatic_Site_Type**[], the site type is defined as that of **Electrostatic_Site_Type**[]. This is necessary to use field electrostatic method.

- **ES_Element** ··· Electric charge/dipole of the electrostatic site
  In the case of **Type_Name** is **POINT_CHARGE**, **ES_Element** is a point charge and in the case of **Type_Name** is **DIPOLE**, it is a dipole moment.

- **atom**[]$\cdots$ Array of index of atom to define the electrostatic site
  When the type of site is **POINT_CHARGE**, the number of element is 1. When the type of site is **DIPOLE**, the number of element is 2. Since the target atom should exist in the same molecule, only index(es) of array **atom**[] are specified.

5.2.8 Structure

Structure has the following hierarchical structure.

```
Structure---Position---mol[]---atom[]
          |-Velocity---mol[]---atom[]
          |-Force   ---mol[]---atom[]
          |-Unit_Cell
```

- **Position** $\cdots$ Position of atoms
- **Velocity** $\cdots$ Velocity of atoms
- **Force** $\cdots$ Force acting on atoms

  mol[].atom[] below **Position**, **Velocity** and **Force** correspond to **Set_of_Molecules.molecule[].atom**[] and specified by the same array index.

- **Unit_Cell** $\cdots$ Information of the unit cell of systems
  - **Density** $\cdots$ Density
  - **Shear_Strain** $\cdots$ Amount of a shear strain
    If **Shear_Strain** is not 0, the Lees-Edwards boundary conditions are applied to the system.
  - **Cell_Size** $\cdots$ Dimension of the unit cell **a, b, c, alpha, beta, gamma**

5.2.9 Unit_Parameter

Unit_Parameter has the following parameters.

- **Name** $\cdots$ Name of unit set
  Arbitrary name can be specified.
- **Comment** $\cdots$ Comment for the unit set
- **Mass** $\cdots$ Reduced mass
  The original data has a unit [amu](atomic mass unit). It can be converted to arbitrary unit in GOURMET.
- **Energy** $\cdots$ Reduced energy
  The original data has a unit [kJ/mol]. It can be converted to arbitrary unit in GOURMET.
- **Length** $\cdots$ Reduced length
  The original data has a unit [nm]. It can be converted to arbitrary unit in GOURMET.

5.2.10 Draw_Attributes

Draw_Attributes has the following parameters.

- **Atom_Type**[] $\cdots$ An array of **Atom_Type** to define attributes
  - **Name** $\cdots$ Name of **Atom_Type**
  - **color** $\cdots$ Color of atom. It is selected from the list.
  - **transparency** $\cdots$ Transparency. (0.0:Full transparency – 1.0:No transparency)

  - **radius** ··· Radius of atom drawn by ball

- **Bond_Potential**[] ··· An array of **Bond_Potential** to define attributes
  - **Name** ··· Name of **Bond_Potential**
  - **color** ··· Color of bond. It is selected from the list.
  - **transparency** ··· Transparency. (0.0:Full transparency – 1.0:No transparency)
  - **radius** ··· Radius of bond drawn by stick/rod

- **Molecule**[] ··· An array of **Molecule** to define attributes
  - **Name** ··· Name of **Molecule**
  - **color** ··· Color of molecule. It is selected from the list.
  - **transparency** ··· Transparency. (0.0:Full transparency – 1.0:No transparency)
  - **radius** ··· Radius of atom drawn by ball-stick

5.3 Description of output UDF

The items in output UDF is the following.

- Input parameter
  The parameters specified in input UDF is output to **Initial** record.

- **Statistics_Data**

  A list of contents in **Statistics_Data** is shown in Table 5.10 . Refer to the text for the details of each parameter.

  The instantaneous value (**Instantaneous**), batch(section) average (**Batch_Average**) and total averages (**Total_Average**) of the following amounts are output to **Statistics_Data**. On the other hand, batch(section) summation and total summation are output for **Energy_Flow**. (Refer to §2.3.5)

  - **Energy** ··· Energy
    Each term of energy is output, i.e.
    Hamiltonian*1 / Total Energy / Kinetic Energy / Potential Energy / Bond Energy / Angle Energy / Torsion Energy / Non Bonding Energy / Coulomb Energy / External Energy

    *1 Total energy is output when the algorithm for the simulation doesn't define the Hamiltonian.

  - **Temperature** ··· Temperature
  - **Pressure** ··· Pressure
  - **Stress**
    - **Total** ··· Normal stress tensor
    - **Bond** ··· Stress tensor obtained from bond term
    - **Non_Bond** ··· Stress tensor obtained from non bond term
  - **Volume** ··· Volume
  - **Density** ··· Density
  - **Cell** ··· Cell size
  - **Wall_Press** ··· Pressure acting on a wall (if the potential of a wall exists)
  - **Energy_Flow** ··· Energy flow by velocity scaling

- **Set_of_Molecules**
  When the simulation with chemical reaction is performed, **Set_of_Molecules** is output to record data after first reaction occurred. Otherwise, it is only output in **Initial** record.

- **Structure**
  Structure is output to every record when the items in **Simulation_Condisitons.Output_Flags.Structure** are set **TRUE**.
  The structure of the object is the same as that of input.

- **Averaged_Structure**
  Averaged_Structure is output to every record when the items in **Simulation_Condisitons.Output_Flags.Averaged_Structure** are set **TRUE**.
  The structure of the object is the same as that of **Structure** except **Unit_Cell**.

- **Grid_Density**
  When the output of density distribution is set up by **Simulation_Conditions.Density_Output**, the density distribution is output to record data.
  The format is similar to that of **SUSHI**. Refer to "SUSHI User's manual" for details.

Table 5.10: Statistics_Data

UDF path	Description
"Energy"	Kinetic and potential energy
"Energy.Instantaneous"	Instantaneous value
"Energy.Batch_Average"	Batch(section) average
"Energy.Total_Average"	Total average
"Temperature"	
"Temperature.Instantaneous"	Instantaneous value
"Temperature.Batch_Average"	Batch(section) average
"Temperature.Total_Average"	Total average
"Density"	
"Density.Instantaneous"	Instantaneous value
"Density.Batch_Average"	Batch(section) average
"Density.Total_Average"	Total average
"Volume"	
"Volume.Instantaneous"	Instantaneous value
"Volume.Batch_Average"	Batch(section) average
"Volume.Total_Average"	Total average
"Cell_Size"	
"Cell_Size.Instantaneous"	Instantaneous value
"Cell_Size.Batch_Average"	Batch(section) average
"Cell_Size.Total_Average"	Total average
"Pressure"	
"Pressure.Instantaneous"	Instantaneous value
"Pressure.Batch_Average"	Batch(section) average
"Pressure.Total_Average"	Total average
"Stress"	Stress tensor
"Stress.Total.Instantaneous"	Total stress, e.g. Bond, Non bond and Kinetic term. Instantaneous value
"Stress.Total.Batch_Average"	Total stress, e.g. Bond, Non bond and Kinetic term. Batch(section) average
"Stress.Total.Total_Average"	Total stress, e.g. Bond, Non bond and Kinetic term. Total average
"Stress.Bond.Instantaneous"	Bond stress. Instantaneous value
"Stress.Bond.Batch_Average"	Bond stress. Batch(section) average
"Stress.Bond.Total_Average"	Bond stress. Total average
"Stress.Non_Bond.Instantaneous"	Non bond stress. Instantaneous value
"Stress.Non_Bond.Batch_Average"	Non bond stress. Batch(section) average
"Stress.Non_Bond.Total_Average"	Non bond stress. Total average
"Wall_Pressure"	Pressure acting on a wall (if wall potential exists)
"Wall_Pressure.Instantaneous"	Instantaneous value
"Wall_Pressure.Batch_Average"	Batch(section) average
"Wall_Pressure.Total_Average"	Total average
"Energy_Flow"	Energy flow by velocity scaling
"Energy_Flow.Batch_Sum"	Batch(section) summation
"Energy_Flow.Total_Sum"	Total summation

- **mesh** $\cdots$ Definition of mesh data. The format is the same as that of regular mesh of **SUSHI**.
- **boundary_conditions** $\cdots$ Definition of boundary conditions. The format is the same as that of **SUSHI**.
- **atom_name**[] $\cdots$ An array of **Atom_Name** corresponding to the density field in **phi.value**[]**.comp**[]
- **phi** $\cdots$ Density (or volume fraction) field. The format is the same as that of regular mesh of **SUSHI**.

- **Correlation_Functions**
  When the output of auto correlation functions is set up by **Simulation_Conditions.Output_Flags.Correlation_Fu**
  the results is output in **the final record**.
  - **Stress** $\cdots$ Auto correlation of stress
    the stress componemt, **xy,yz,zx,xx-yy,yy-zz** and the relaxation modulus, **G_t** obtained by the Green-Kubo fomula are output as a function of **Time**.
- **Unit_Parameter** $\cdots$ Scaling parameter for unit conversion
  This is the same structure as that in input UDF.

5.4 Description of table UDF

Table UDF has data for table potentials for bond, angle, torsion and non-bonding potentials. The data definition of this table UDF partially follows the definition of Mesh & Field.

- **Mesh**: Information of mesh

  In the case of table potential, the position of mesh corresponds to bond length, angle and unit width, for examples. Parameters of **Mesh** are described in Table 5.11. Refer to the text for the details of each parameter.

Table 5.11: Mesh

UDF path name	Description
"name"	Name of mesh
"type"	Type of mesh (not used in **COGNAC**)
"axes[]"	Data of axis (The number of element of the array corresponds to the dimension of mesh)
"axes[*].values"	Parameters for the size and the number of mesh

  - **name** $\cdots$ Name of mesh
  - **type** $\cdots$ Type of mesh
    Since **COGNAC** supports only regular mesh, this parameter is not used for the specification of mesh type.

    Exception: When the **type** is specified as **"FORCE"**, the tabular data is interpreted not as potential energy but as force values derived from potentials.

  - **axes[]** : Data for axis
    The number of element of array corresponds to the dimension of mesh.
    - **values[]** : Data for the size and the number of mesh on each axis.
      The following data is specified for a regular mesh.
      values[0] $\cdots$ Minimum value
      values[1] $\cdots$ Maximum value
      values[2] $\cdots$ Number of division

      In the case of table potentials, if this parameter is set as 100, 101 data points are need including both ends of the division.

- **FieldValue**: Information of field value on mesh

  In the case of table potential, the value usually corresponds to potential energy.
  The parameters of **FieldValue** is described in Table 5.12. Refer to the text for the details of each parameter.

Table 5.12: FieldValue

UDF path name	Description
"value[]"	Array of the data on mesh

  - **value**[]$\cdots$ Array of the data on mesh

- **Unit_Parameter**: Scaling parameter for unit conversion
  This is the same structure as that in input/output UDF.

5.5 Description of crystal UDF

Crystal UDF has the following data.

- **Crystal_Data**: Information of lattice constants.

  The parameters of **Crystal_Data** is described in Table 5.13. Refer to the text for the details of each parameter.

Table 5.13: Crystal_Data

UDF path name	meaning
"Unit_Cell"	Information of unit lattice of crystal
"Unit_Cell.a"	length of a axis
"Unit_Cell.b"	length of b axis
"Unit_Cell.c"	length of c axis
"Unit_Cell.alpha"	angle α
"Unit_Cell.beta"	angle β
"Unit_Cell.gamma"	angle γ
"Symmetry_Operation"	Information of symmetrical operation
"Symmetry_Operation.Operation[]"	Array of symmetrical operations (translation and inversion)
"Fractional_Coordinate"	Information of fractional coordinates
"Fractional_Coordinate.Coordinate[]"	Array of fractional coordinates

  The parameters of **Crystal_Data** is explained below.

  - **Unit_Cell** $\cdots$ Information of unit lattice of crystal
  - **Symmetry_Operation.Operation[]** $\cdots$ Array of symmetrical operations
    The array includes elements of the number of symmetrical operations.

    * **Invert_Vector** $\cdots$ Vector of inverse operation
    * **Trans_Vector** $\cdots$ Vector of translational operation

      A Fractional coordinate **R** of asymmetric unit is converted into **Invert_Vector**:**R**+**Trans_Vector** using the data of **Operation[]**.

  - **Fractional_Coordinate.Coordinate[]** $\cdots$ Array of fractional coordinates in an asymmetric unit
    * **x**,**y**,**z** $\cdots$ Components of fractional coordinates in an asymmetric unit

- **Unit_Parameter**: Scaling parameter for unit conversion
  This is the same structure as that in input/output UDF.

Chapter 6

–SILK– The tool for COGNAC input UDF creation

In this chapter, how to use **SILK** (the tool for **COGNAC** input UDF creation) is explained.

SILK(Set of molecules Interpreter of Light Kits) is the tool of creating a part of data for **COGNAC** input, **Set_of_Molecules**.

Set_of_Molecules is the data structure which defines the topology of molecules.

It is hard to edit its contents directly, especially in the case of modeling polymers consisting of large amount of atoms. However, users can reduce the task with using **SILK**.

In this chapter, an simple example of creating **Set_of_Molecules** by using **ACTION** is introduced first. And next is some procedures to create **Set_of_Molecules** by **Python** on **GOURMET**.

6.1 Start up of SILK : Creation of COGNAC input UDF files by ACTION

A procedure to create **COGNAC** input UDF files using **ACTION** is explained.

1. Potential_Map

   A **SILK** input UDF data has almost the same structure as the **COGNAC** input UDF data. The template UDF file "potential_map.udf" is prepared, and users can edit and use it for the **SILK** input UDF data. Start **GOURMET** and load the "potential_map.udf". In a **GOURMET** window, the data of record number **0** is displayed. Move the **slide bar** at the lower part in the **GOURMET** window to the right, and check that **Record Label**, which is displayed at a under-right position in the window, shows "ACTION_SILK_TEST" (record number **12**).

2. construction of molecular architecture

   On the left aria in **GOURMET** window, the tree like structure of the data is displayed. The new window is pop up when the right button of a mouse is clicked on the label **Set_of_Molecules**.

   Followings are menu.

   - **SILK_CREATE_CombPolymer_Bead_Spring_2_branched**
     This creates comb polymers with two branching points (bead-spring model) , e.g. **A10(B5)-A10(B5)-A10**.

   - **SILK_CREATE_CombPolymer_Bead_Spring_X_branched**
     This creates comb polymers of arbitrary number, length and atom type of blocks (bead-spring model), e.g. **A10(B5)-A5(C5)-A7(B3)-A10**.

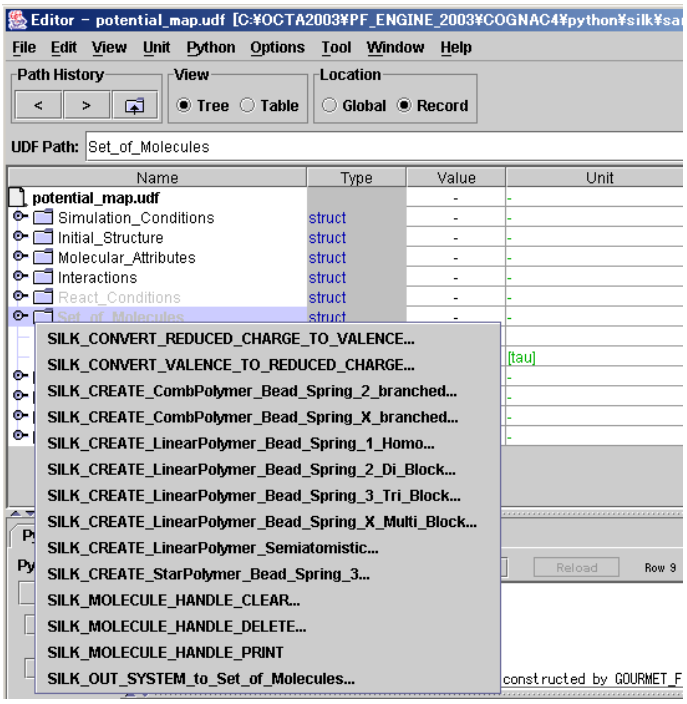

Figure 6.1: Starting of **ACTION**

- **SILK_CREATE_LinearPolymer_Bead_Spring_1_Homo**
  This creates linear polymers (bead-spring model), e.g. **A10**.

- **SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block**
  This creates linear di-block copolymers (bead-spring model), e.g. **A10-B10**.

- **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block**
  This creates linear tri-block copolymers (bead-spring model), e.g. **A10-B10-C10**.

- **SILK_CREATE_LinearPolymer_Bead_Spring_X_Multi_Block**
  This creates arbitrary number of linear multi-block copolymers (bead-spring model), e.g. **A20-B40-A20-C10**.

- **SILK_CREATE_LinearPolymer_Semiatomistic**
  This creates linear polymers (united atom model).

- **SILK_CREATE_Monomer**
  This creates monoatomic molecules.

- **SILK_CREATE_StarPolymer_Bead_Spring_3**
  This creates star polymers with three arms (bead-spring model).

Some messages are displayed on **Python Log** panel when a function is chosen and executed from the menu.

When

SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block is executed, following are shown on **Python Log** panel.

```
Trying to constructing LINEAR_Block_molecules...
[('A', 10), ('B', 10)]
Section of constructing LINEAR_Block_Polymers. Done.
```

Any other function in the menu can be done successively.

Follwing are shown on **Python Log** panel when

SILK_CREATE_LinearPolymer_Bead_Spring_1_Homo is chosen.

```
Trying to constructing LINEAR_Homo_molecules...
[('A', 10)]
Section of constructing LINEAR_Block_Polymers. Done.
```

When users want to confirm the constructed molecules,

SILK_MOLECULE_HANDLE_PRINT in the menu is covenient.

Follwing are shown on **Python Log** panel.

```
============Registered Molecules==================
item_ 0  Name of Molecule: linear_di  Num of Molecule: 1
item_ 1  Name of Molecule: linear_homo  Num of Molecule: 1
==================================================
```

3. Write data of constructed molecules to **Set_of_Molecules**

   SILK_OUT_SYSTEM_to_Set_of_Molecules in the menu is the command which writes the data of constructed molecules to **GOURMET** worksheet (**Set_of_Molecules**)

   Followings are messages on execution.

   ```
   try_to_write_Set_of_Molecules ...
   item(MOL_NAME,NUM_OF_MOL) [('linear_di', 1), ('linear_homo', 1)]
   ```

   On **GOURMET** worksheet(**Set_of_Molecules**), constructed molecules are written in **Set_of_Molecules**.

4. Creation of UDF file(**COGNAC** input file)

   On the left aria in **GOURMET** window, the tree like structure of the data is displayed. The new window is pop up when the right button of a mouse is clicked on the label of **file name**(at the top of tree like structure).

   The small windows is pop up when user chooses **SILK_UDF_CREATE**. The dialog window is pop up when clicking a right button of a mouse in **values** column.

6.2 Outline of creation of COGNAC input UDF files

A procedure to create **COGNAC** input UDF files using **SILK** basic functions is explained.

1. Edit of **SILK** input UDF data

   A **SILK** input UDF data has almost the same structure as the **COGNAC** input UDF data. The template UDF file "potential_map.udf" is prepared, and users can edit and use it for the **SILK** input UDF data.

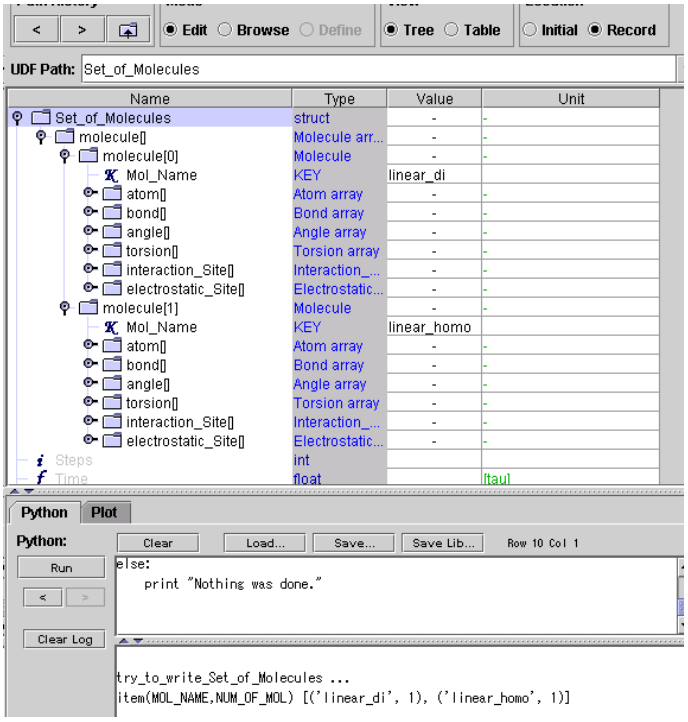

Figure 6.2: An example of executing **SILK_OUT_SYSTEM_to_Set_of_Molecules"**

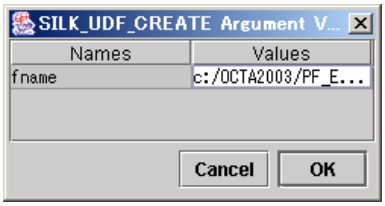

Figure 6.3: Dialog window for creating UDF file

2. Load of user editable Python script

   SILK consists of some scripts written in Python("*.py" file).

   A specific script among them ("silk_use_*.py") is loaded on **GOURMET**.

3. Edit of user editable Python script

   In the loaded script ("silk_use_*.py"), there are some lines which user have to edit, i.e. a location of output file, a modeling procedure of polymer, etc.

   SILK users can save the edited one as arbitrary file name and load it again on **GOURMET**.

4. Execution of the Python script (user editable Python script).

   At the execution of the Python script on **GOURMET**, **SILK** creates the **COGNAC** input file.

   In **GOURMET**, the log of the execution is displayed on the **Python panel** in the **GOURMET** window.

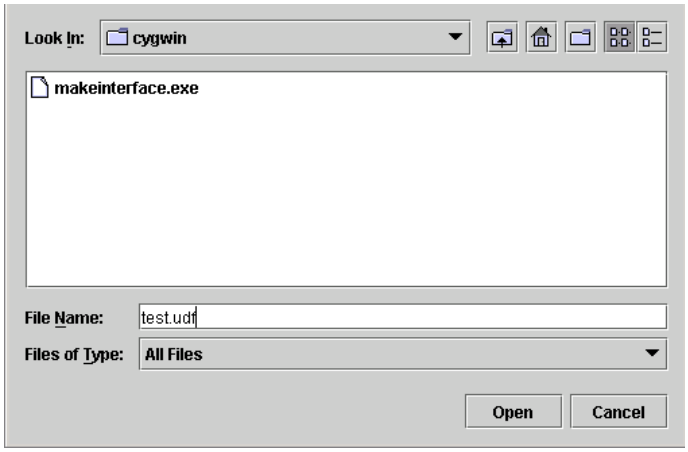

Figure 6.4: The window for setting file name

6.2.1 Preparation of SILK execution (edit of Potential_Map)

In order to execute **SILK**, the UDF("potential_map.udf" is a template file, and the file can have an arbitrary name.), in which the simulation conditions(temperature, pressure etc.) of the **COGNAC** must be included. This **SILK** input UDF data has the almost same structure as the **COGNAC** input UDF data. **SILK** user have to edit parameters in the UDF data, such as potential parameters, temperature and pressure etc., for the simulation. The operation is explained below using "potential_map.udf" as a template.

- Load of **SILK** input UDF

  Start **GOURMET** and load the **SILK** input UDF file ("potential_map.udf"). In a **GOURMET** window, the data of record number **0** is displayed. Move the **slide bar** at the lower part in the **GOURMET** window to the right, and check that **Record Label**, which is displayed at a under-right position in the window, shows "UA_PE_Kuwajima" (record number **2**).

  On the left side of **GOURMET** window, the structure of the data is displayed. Each data at the current record can be browsed by clicking of the icons of the structure.

- Edit of UDF

  SILK users can edit the data on **GOURMET**.

  - **Molecular_Attributes**
    Setup of the atomic type, interaction site and bonding potential
  - **Interactions**
    Setup of the non-bonding interaction, Electrostatic potential and external potential
  - **Unit_Parameter**
    Scaling parameters for unit conversion Since this is not required for the **COGNAC** execution, this structure can be kept blank.

    By defining these units, other units such as temperature and pressure can be converted. The script named "silk_mujigenkun.py" can be loaded and used to convert units.
  - **Initial_Structure**
    Setup of the initial structure, e.g. random, helix, crystal and restart
  - **Simulation_Conditions**
    Setup of the general simulation condition, e.g. temperature, pressure, time step and ensemble

- Save of input UDF data

  After each parameter is edited, the input UDF file should be updated from the menu on **GOURMET**.

6.2.2 Polymer modeling by SILK (edit of user editable Python script)

- Load of Python script

  Load "silk_use.py" to **GOURMET** and confirm that the script is displayed on the **Python panel** in **GOURMET** window. As it is shown in the next step, edit the upper part of the script. The following items must be edited.

  – Output location
  – Modeling of molecules

- Edit of the Python script

  The section (**SECTION "USER DEFINITION"**) which **SILK** user should edit is in the upper part of the loaded script.

  – Output location
  SUBSECTION "outputpath" of **"USER DEFINITION"** is explained below.

```
####### SUBSECTION "outputpath" ######
def setOutParam(self):
#output Directory (ex. outDir="c:/OCTA/***" (dos), outDir="/home/yourdir/***")
self.engine.outDir="C:/OCTA/PF_ENGINE_2013/COGNAC/python/silk/sample"
#filename without suffix(.udf)
self.engine.cognacFileName="test_in"
#project name
self.engine.prjName="DEVELOP"
#output file is diveded to Structure_data and other Parameters when "TWO_FILES" is chosen.
self.engine.fileNumCom="ONE_FILE"
#self.engine.fileNumCom="TWO_FILES"
```

  The directory of output location is specified in **self.engine.outDir**.
  Notice that slash(/) will work to specify the directory path even in the Windows(TM) environment.

  The output file name without UDF suffix ".udf" is specified in **self.engine.cognacFileName**.

  The project name of the output file is specified in **self.engine.prjName**.

  The flag of whether to create one file or two files for the output of **SILK** (input of **COGNAC**) is specified in **self.engine.fileNumCom**. When **self.engine.fileNumCom** is specified as **ONE_FILE**, the input data for **COGNAC** is output into a single file .
  When **self.engine.fileNumCom** is specified as **TWO_FILE**, the input data for **COGNAC** is output into two files. One file has a topological data (**Set_of_Molecules**) and the other file has the rest of input data.

  – Generation of molecules
  The contents of **SUBSECTION "system"** in **"USER DEFINITION"** are explained. Confirm that the following text is in the loaded script.

```
#######  SUBSECTION "system"  ######
def userDef(self):
```

```
##### BuildSytem(Make sure to execute at "record 6") #####
name="mol"
numMol=64
self.engine.createMolecule(name)
for i in range(0, 4):
self.engine.addAtoms(name, "UA", "UA_Kuwajima")
for i in range(0, 3):
self.engine.addBonds(name,  i, i+1, "BOND_PE_Kuwajima")
for i in range(0, 2):
self.engine.addAngles(name, i, i+1, i+2, "ANGLE_PE_Kuwajima")
for i in range(0, 1):
self.engine.addTorsions(name, i, i+1, i+2, i+3, "TORSION_PE_Kuwajima")
for i in range(0, 4):
self.engine.addInteractionSites(name, [i], "NB_PE_Kuwajima", "PAIR")
self.engine.setSystem(name, numMol)
```

1. Registration of molecules
   A name of molecules should be registered first. **"mol"** is the name of molecules.

   ```
   name="mol"
   ```

   numMol is a number of molecules.

   ```
   numMol=10
   ```

   The following function registers molecules to **SILK**. **"self.engine."** is an indispensable description.

   ```
   self.engine.createMolecule(name)
   ```

2. Registration of atom(s) to the molecule (molecular name : **"mol"**)

   ```
   self.engine.addAtoms(name, "UA", "UA_Kuwajima")
   ```

   This function **"addAtoms(...)"** registers an atom. **"self.engine."** is an indispensable description.

   The 1st argument(**name**) is a name of the molecule to register an atom (**"mol"** is this example).

   The 2nd argument(**"UA"**) is a name of the atom. Users can set arbitrary name.

   The 3rd argument(**"UA_Kuwajima"**) is a type of the atom.

   An example which registers four atoms to the molecule **"mol"** is following. **"UA"** is the name of atoms, **"UA_Kuwajima"** is the type of atoms.

   Four atoms are sequentially registered.

   ```
   for i in range(0,4):
     self.engine.addAtoms(name, "UA", "UA_Kuwajima")
   ```

3. Registration of bond(s) to the molecule (molecular name : **"mol"**)

```
self.engine.addBonds(name,0,1, "BOND_PE_Kuwajima")
```

This function **"addBonds(...)"** registers a bond.
"self.engine." is an indispensable description.

The 1st argument is a name of the molecule to register an atom (**"mol"** is this example).

The 2nd argument (**0**) is the index of atom which forms the bond.

The 3rd argument (**1**) is the index of atom which forms the bond.

The 4th argument (**"BOND_PE_Kuwajima"**) is a potential name of the bond.

An example which registers three bonds to the molecule **"mol"** is following. The name of bond potential is **"BOND_PE_Kuwajima"**)

Three bonds are sequentially registered.

```
for i in range(0,3):
  self.engine.addBonds(name, i, i+1, "BOND_PE_Kuwajima")
```

4. Registration of angle(s) to the molecule (molecular name : **"mol"**)

```
self.engine.addAngles(name,0,1,2, "ANGLE_PE_Kuwajima")
```

This function **"addAngles(...)"** registers an angle.
"self.engine." is an indispensable description.

The 1st argument is a name of the molecule to register an atom (**"mol"** is this example).

The 2nd argument (**0**) is the index of atom which forms angle.

The 3rd argument (**1**) is the index of atom which forms angle.

The 4th argument (**2**) is the index of atom which forms angle.

The 5th argument (**"ANGLE_PE_Kuwajima"**) is a potential name of the angle.

An example which registers two angles to the molecule **"mol"** is following. The name of angle potential is **"Angle_PE_Kuwajima"**).

Two angles are sequentially registered.

```
for i in range(0,2):
  self.engine.addAngles(name, i, i+1, i+2, "ANGLE_PE_Kuwajima")
```

5. Registration of torsion(s) to the molecule (molecular name : **"mol"**)

```
self.engine.addTorsions(name,0,1,2,3, "TORSION_PE_Kuwajima")
```

This function **"addTorsions(...)"** registers a torsion.
"self.engine." is an indispensable description.

The 1st argument (**name**) is a name of the molecule to register an atom (**"mol"** is this example).

The 2nd argument (**0**) is the index of atom which forms torsion.

The 3rd argument (**1**) is the index of atom which forms torsion.

The 4th argument (**2**) is the index of atom which forms torsion.

The 5th argument (**3**) is the index of atom which forms torsion.

The 6th argument (**"TORSION_PE_Kuwajima"**) is a potential name of the torsion.

An example which registers one torsion to the molecule named **"mol"** is following. (the type of torsion is **"TORSION_PE_Kuwajima"**)

One torsion is (sequentially) registered.

```
for i in range(0,1):
  self.engine.addTorsions(name, i, i+1, i+2, i+3,"TORSION_PE_Kuwajima")
```

6. Registration of interaction site(s) to the molecule (molecular name : **"mol"**)

```
self.engine.addInteractionSites(name, [0], "NB_PE_Kuwajima", "PAIR")
```

This function **"addInteractionSites(...)"** registers an interaction site for non-bonding interaction.

The 1st argument (**name**) is a name of the molecule to register an atom (**"mol"** is this example).

The 2nd argument(**[0]**) is an array of the index of atom.

Since interaction sites can be defined by multi atoms, this argument is specified by an array.

The 3rd argument(**"NB_PE_Kuwajima"**) is the type of the interaction site.

The 4th argument(**"PAIR"**) means this site acts as the site for non-bonding pair interaction (e.g. Lennard-Jones interaction).

7. Setup for the output of the registered molecule (molecular name : **"mol"**)

```
self.engine.setSystem(name, numMol)
```

Here, the number of the molecules((name : **"mol"**) is set.

numMol is the number of the molecules in a system.

Notice that the data is not output unless the function **setSystem(...)** is called even if the molecular data is registered in **SILK**.

- Execution of the Python script

  After the contents (output place etc.) of the edited Python script are confirmed, run the script. A result of execution will be displayed in a log window.

6.3 Functions

6.3.1 Location of the output

Output location (directory and filename) is set by the function **"setOutParam()"**. The function **"setOutParam()"** is placed in the upper part of the sample file such as "silk_use.py".

- **self.engine.outDir** ··· Directory of the output location

  An example is shown below.

```
self.engine.outDir="C:/OCTA8.3/ENGINES/COGNAC/python/silk/sample"
```

- **self.engine.cognacFileName** ··· File name of output UDF without extension ".udf"

  SILK output UDF file is the input UDF file of **COGNAC**. An example is shown below.

```
self.engine.cognacFileName="peo"
```

  "peo.udf" is set as the output file name.

- **self.engine.prjName** ··· Project name of the output file

  An example is shown below.

```
self.engine.prjName="DEVELOP"
```

  DEVELOP is set as the project name of the output file.

- **self.engine.fileNumCom** ··· Type of output

  Examples are shown below.

```
self.engine.fileNumCom="ONE_FILE"
```

  In this case, all parameters are output to "peo.udf".

```
self.engine.fileNumCom="TWO_FILES"
```

  In this case, the topological data of molecules is output to "peo_str.udf" while other parameters are output to "peo.udf".

6.3.2 Registration of molecules

```
self.engine.createMolecule(name)
```

The argument (**name**) ··· Name of the molecule

6.3.3 Registration of atoms to molecules

```
self.engine.addAtoms(name, AtomName, AtomTypeName)
```

The 1st argument (**name**) ··· Name of the molecule to register an atom

The 2nd argument (**AtomName**) ··· Name of the atom. Arbitrary name can be set

The 3rd argument (**AtomTypeName**) ··· Type of the atom

- Note on the 3rd argument

  An array of UDF path name, **"Molecular_Attributes.Atom_Type[]"** is referred when **COGNAC** executes. More to say, this parameter should agree with **Name** of one element in the array.

6.3.4 Setup of the various parameters for atoms

```
self.engine.setAtomParam(molName, seq, paramName, param)
```

The 1st argument (**molName**) ··· Name of the molecule which consists of the atom
The 2nd argument (**seq**) ··· Index of the atom in the molecule
The 3rd argument (**paramName**) ··· Selection of the parameter to set the value(keyword)
The 4th argument (**param**) ··· Value of the parameter

- Keyword of the 3rd argument and value

  "Chirality" ··· **0** or **1**

  "Main_Chain" ··· **0** or **1**

  "Atom_ID" ··· atom ID(arbitrary integer)

6.3.5 Registration of bonds to molecules

```
self.engine.addBonds(name, atom1, atom2, BondTypeName)
```

The 1st argument (**name**) ··· Name of the molecule to register a bond
The 2nd argument (**atom1**) ··· Index of an atom which forms the bond
The 3rd argument (**atom2**) ··· Index of an atom which forms the bond
The 4th argument (**BondTypeName**) ··· Type of the bond

- Note on the 4th argument

  An array of UDF path name, **"Molecular_Attributes.Bond_Potential[]"** is referred when **COGNAC** executes. More to say, this parameter should agree with the **Name** of one element in the array.

6.3.6 Registration of angles to molecules

```
self.engine.addAngles(name, atom1, atom2, atom3, AngleTypeName)
```

The 1st argument (**name**) ··· Name of the molecule to register an angle
The 2nd argument (**atom1**) ··· Index of an atom which forms the angle
The 3rd argument (**atom2**) ··· Index of an atom which forms the angle
The 4th argument (**atom3**) ··· Index of an atom which forms the angle
The 5th argument (**AngleTypeName**) ··· Type of the angle

- Note on the 5th argument

  An array of UDF path name, **"Molecular_Attributes.Angle_Potential[]"** is referred when **COGNAC** executes. More to say, this parameter should agree with **Name** of one element in the array.

6.3.7 Registration of torsions to molecules

```
self.engine.addTorsions(name, atom1, atom2, atom3, atom4, TorsionTypeName)
```

The 1st argument (**name**) ··· Name of the molecule to register a torsion
The 2nd argument (**atom1**) ··· Index of an atom which forms the torsion
The 3rd argument (**atom2**) ··· Index of an atom which forms the torsion
The 4th argument (**atom3**) ··· Index of an atom which forms the torsion
The 5th argument (**atom4**) ··· Index of an atom which forms the torsion
The 6th argument (**TorsionTypeName**) ··· Type of the torsion

- Note on the 6th argument

  An array of UDF path name, **"Molecular_Attributes.Torsion_Potential[]"** is referred when **COGNAC** executes. More to say, this parameter should agree with **Name** of one element in the array.

6.3.8 Registration of interaction sites for pair interactions to molecules

```
self.engine.addInteractionSites(name,[atomID], InteractionSiteTypeName, "PAIR")
```

The 1st argument (**name**) ··· Name of the molecule to register an interaction site
The 2nd argument ([**atomID**]) ··· Array of the index of atom(s)
The 3rd argument (**InteractionSiteTypeName**) ··· Type of the interaction site
The 4th argument (**"PAIR"**) ··· Keyword to specify the site for pair interaction

- Note on the 3rd argument

  An array of UDF path name, **"Molecular_Attributes.Interaction_Site_Type[]"** is referred when **COGNAC** executes. More to say, this parameter should agree with **Name** of one element in the array.

6.3.9 Registration of interaction sites for external fields to molecules

```
self.engine.addInteractionSites(name,[atomID], InteractionSiteTypeName, "EXTERNAL")
```

The 1st argument (**name**) ··· Name of the molecule to register an interaction site
The 2nd argument ([**atomID**]) ··· Array of the index of atom(s)
The 3rd argument (**InteractionSiteTypeName**) ··· Type of the interaction site
The 4th argument (**"EXTERNAL"**) ··· Keyword to specify the site for external field

- Note on the 3rd argument

  An array of UDF path name, **"Molecular_Attributes.Interaction_Site_Type[]"** is referred when **COGNAC** executes. More to say, this parameter should agree with **Name** of one element in the array.

6.3.10 Registration of electrostatic sites to molecules

```
self.engine.addInteractionSites(name,[atomID], ESInteractionName, "COULOMB",ESvalue)
```

The 1st argument (**name**) ··· Name of the molecule to register an electrostatic site
The 2nd argument ([**atomID**]) ··· Array of the index of atom(s)
The 3rd argument (**ESInteractionName**) ··· Name of the type electrostatic interaction
The 4th argument (**"COULOMB"**) ··· Keyword to specify the site for electrostatic interaction
The 5th argument (**ESvalue**) ··· Dipole moment or charge

6.3.11 Specification of the size of system

```
self.engine.setSystem(name, numMol)
```

The 1st argument (**name**) ··· Name of the molecules
The 2nd argument (**numMol**) ··· Number of the molecules

6.4 Template functions for polymer modeling

6.4.1 Linear homo-polymer

This function creates a set of linear homopolymers of coarse-grained model (from united atom to bead spring model) as it is shown in Figure 6.5.

```
self.engine.makeLinPolym(name, numAtom, numMol, atomName, atomTypeName, bondTypeName,
                         angleTypeName, torsionTypeName, interactionSiteTypeName)
```

The 1st argument (**name**) ··· Name of the molecule
The 2nd argument (**numAtom**) ··· Number of atoms in the molecule
The 3rd argument (**numMol**) ··· Number of the molecules
The 4th argument (**atomName**) ··· Name of the atoms
The 5th argument (**atomTypeName**) ··· Type of the atoms
The 6th argument (**bondTypeName**) ··· Potential name of the bonds
The 7th argument (**angleTypeName**) ··· Potential name of the angles
The 8th argument (**torsionTypeName**) ··· Potential name of the torsions
The 9th argument (**interactionSiteTypeName**) ··· Type of the interaction sites

- Note on the 7th argument

  Angle is not set when the argument is set to **""**

- Note on the 8th argument

  Torsion is not set when the argument is set to **""**

- Note on the 9th argument

  Interaction site is not set when the argument is set to **""**

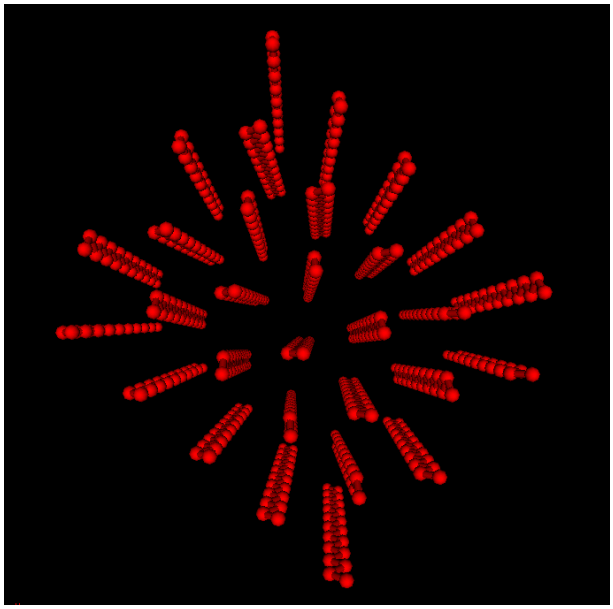

Figure 6.5: Alkanes created by **SILK**

6.4.2 Linear homo-polymer II (with molecular weight distribution)

This function creates a set of linear homopolymers of coarse-grained model (from united atom to bead spring model) with molecular weight distribution. The information of molecular weight distributions is taken from the output file of FORK (pre-processor of PASTA).

```
self.engine.makeLinPolymMWD(mwdFileName, baseName, numBeadInZ, chainNumScale,
                            atomName, atomTypeName, bondTypeName, angleTypeName,
                            torsionTypeName, interactionSiteTypeName)
```

The 1st argument (**mwdFileName**) ··· Name of the FORK output file
The 2nd argument (**baseName**) ··· Basic name of the molecules
The 3rd argument (**numBeadInZ**) ··· Number of atoms in a unit Z
(Z corresponds to molecular weight between entanglement)
The 4th argument (**chainNumScale**) ··· Scale ratio of the number of molecules
The 5th argument (**atomName**) ··· Name of the atoms
The 6th argument (**atomTypeName**) ··· Type of the atoms
The 7th argument (**bondTypeName**) ··· Type of the bonds
The 8th argument (**angleTypeName**) ··· Type of the angles
The 9th argument (**torsionTypeName**) ··· Type of the torsions
The 10th argument (**interactionSiteTypeName**) ··· Type of the interaction sites

- Note on the 2nd argument

  For example, when the argument is set to **"mol"**, **SILK** sets the name of each molecule as **"mol_1"** and **"mol_2"** ...

- Note on the 8th argument

  Angle is not set when the argument is set to **""**

- Note on the 9th argument

  Torsion is not set when the argument is set to **""**

- Note on the 10th argument

  Interaction site is not set when the argument is set to **""**

6.4.3 Linear multiblock copolymer (bead spring model)

This function creates a set of linear multi-block polymers of bead spring model as it is shown in Figure 6.6

```
self.engine.makeBeadSpringPolym(name, numMol, "LINEAR", sequence,
                                atomType, bondType, interactionSiteType)
```

The 1st argument (**name**) ··· Name of the molecules
The 2nd argument (**numMol**) ··· Number of the molecules
The 3rd argument (**"LINEAR"**) ··· Keyword. **"LINEAR"** must be specified to model linear polymers.
The 4th argument (**sequence**) ··· Block sequence
The 5th argument (**atomType**) ··· Relation between the atom and its type
The 6th argument (**bondType**) ··· Relation between the atoms in sequence defining bond and its type
The 7th argument (**interactionSiteType**) ··· Relation between the atom in sequence defining interaction site and its type

- Note on the 4th argument

  In the case of modeling the triblock copolymer of A20-B40-A20, the argument is described as follows

  [("A", 20), ("B", 40), ("A", 20)]

- Note on the 5th argument

  In the case that the argument is described as **{"A":"atom1", "B":"atom2"}**, **"atom1"** and **"atom2"** are registered as atom types of atoms of name **"A"** and **"B"** respectively.

- Note on the 6th argument

  In the case that the argument is described as **{"A_A":" bond1", "A_B":" bond3", "B_B":"bond2"}**,
  the type of bond formed by atom **"A"** and atom **"A"** is set to **"bond1"**,
  the type of bond formed by atom **"A"** and atom **"B"** is set to **"bond3"** and
  the type of bond formed by atom **"B"** and atom **"B"** is set to **"bond2"**

- Note on the 7th argument

  In the case that the argument is described as **{"A":"siteType1", "B":"siteType2"}**,

  the type of interaction site defined by atom **"A"** is set to **"siteType1"** and **"B"** is set to **"siteType2"**

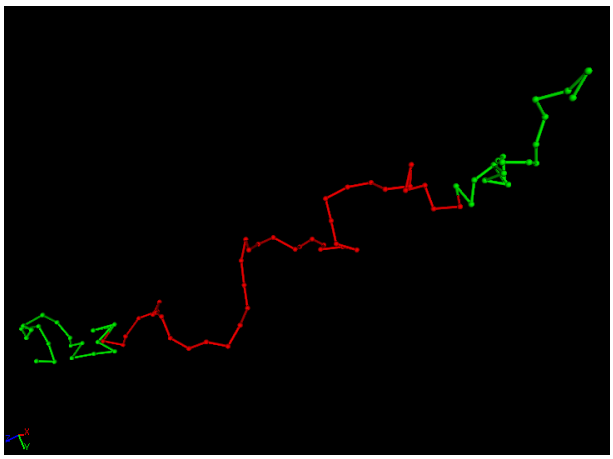

Figure 6.6: Triblock polymer created by **SILK**

6.4.4 Comb polymer (bead spring model)

Any number of comb polymers of bead-spring model can be created by **SILK**, as it is shown in Figure 6.7.
Moreover, it is also possible to create star shaped polymers, as it is shown in Figure 6.8.

```
self.engine.makeBeadSpringPolym(name, numMol, "COMB", sequence,
                                atomType, bondType, interactionSiteType)
```

The 1st argument (**name**) ··· Name of the molecules
The 2nd argument (**numMol**) ··· Number of the molecules
The 3rd argument (**"COMB"**) ··· Keyword. **"COMB"** must be specified to model comb or star polymers.
The 4th argument (**sequence**) ··· Block sequence
The 5th argument (**atomType**) ··· Relation between the atom and its type
The 6th argument (**bondType**) ··· Relation between the atom in sequence defining bond and its type
The 7th argument (**interactionSiteType**) ··· Relation between the atom in sequence defining interaction site and its type

- Note on the 4th argument

  In the case of producing a comb polymer of A20(B20)-A20(B20)-A20 ("B20" is a branched chain), the argument is described as follows:

  [("A", 20), ("B", 20), ("A", 20), ("B", 20), ("A", 20)].

  Moreover, a star shaped (4 armed) polymer is created by describing as **[("A", 20), ("B", 20), ("A", 0), ("B", 20), ("A", 20)]**. In this case, the branched polymer consisting of A40 main-chain and two B20 branching chain is created as shown in Figure 6.8.

- Note on the 5th argument

  In the case that the argument is described as **{"A":"atom1" and "B":"atom2"}**, **"atom1"** and **"atom2"** are registered as atom types of atoms of name **"A"** and **"B"** respectively.

- Note on the 6th argument

  In the case that the argument is described as **{"A_A":" bond1", "A_B":"bond3", "B_B":"bond2"}**,
  the type of bond formed by atom **"A"** and atom **"A"** is set to **"bond1"**,
  the type of bond formed by atom **"A"** and atom **"B"** is set to **"bond3"** and
  the type of bond formed by atom **"B"** and atom **"B"** is set to **"bond2"**

- Note on the 7th argument

  In the case that the argument is described as **{"A":" siteType1", "B":"siteType2"}**,

  the type of interaction site defined by atom **"A"** is set to **"siteType1"** and

  the type of interaction site defined by atom **"B"** is set to **"siteType2"**

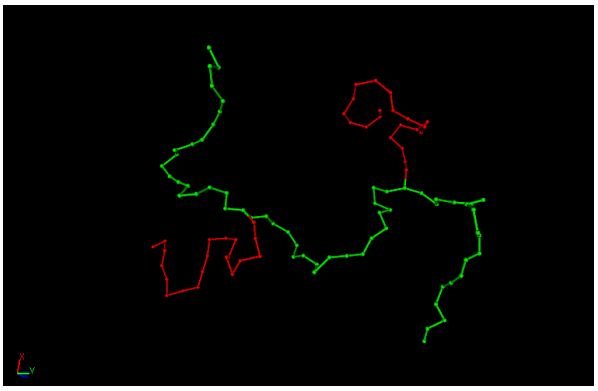

Figure 6.7: Comb polymer created by **SILK** (sequence is [("A",20),("B",20),("A",20),("B",20),("A",20)])

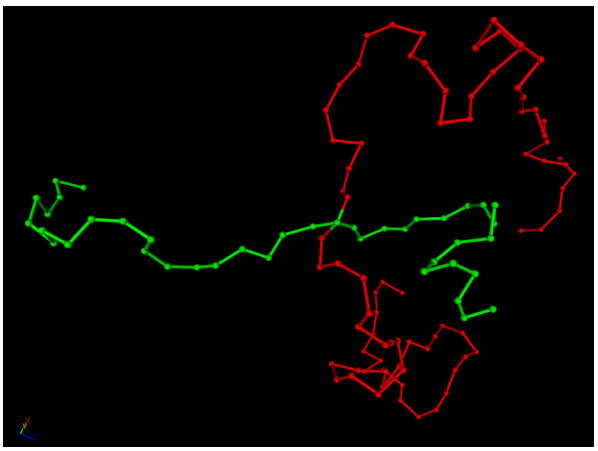

Figure 6.8: Star polymer created by **SILK** (sequence is [("A",20),("B",20),("A",0),("B",20),("A",20)])

Chapter 7

Tools for analysis

Some Python scripts are available to perform analysis, data format conversions, and graphical display of the results of the simulation of **COGNAC**. In addition, some **Action** commands are developed based on the Python scripts. Users can use the functions of the Python scripts from the graphical interface of GOURMET.

This chapter explains the functions of the Python scripts and **Action** commands prepared for **COGNAC**.

All the Python script files and the **Action** files introduced here are placed in t"python/analysis" and "action" directory respectively.

7.1 List of Python script

In order to display and/or analyze the calculation result of **COGNAC**, the following Python script and Library is available.

- "CognacShowLib.py" ··· Display of molecular structure
- "CognacBasicAnalysis.py" ··· Calculation of the basic properties, such as distances and angles
- "CognacGeometryAnalysis.py" ··· Analysis of single record data, such as pair distributions function and density distributions
- "CognacTrajectoryAnalysis.py" ··· Analysis of time dependent data, such as mean square displacements
- "CognacFileConvert.py" ··· Conversion of UDF files to other file formats
- "CognacUtiltity.dll/so" ··· Library for performing the translation of coordinates conversion by boundary conditions, calculating self-correlation functions, etc.
  Input and output are independent from UDF.

Since **CognacBasicAnalysis** is designed as a subclass of class **UDFManager**, the methods of **UDFManager**, such as **get** and **jump**, can be used in addition to the methods explained below. Refer to "GOURMET Python script reference manual" for the detail of the methids of **UDFManager**.

CognacGeometryAnalysis, **CognacTrajectoryAnalysis** and **CognacFileConvert** are subclasses of **CognacBasicAnalysis**, and all the methods of **CognacBasicAnalysis** can also be used.

The return values of the methods in these classes are one of standard Python types, i.e. simple values such as **integer** and **float**, or a sequence **list** and **tuple**.

7.2 Installation

To use the Python scripts, add the directory, where the files are located, to an environment variable **PYTHONPATH**, or copy all scripts/library to the directory, which is already set in the **PYTHONPATH**.

If the GOURMET and simulation programs are installed using the installer, users don't have to do the procedure.

7.3 How to use

The Python scripts are used by importing them in GOURMET or a usual Python command line and script files except "CognacShowLib.py". Since "CognacShowLib.py" is a script for displaying molecular structure on **View** window of GOURMET, it can be used only GOURMET python window.

7.4 Description of classes and methods

7.4.1 CognacShowLib.py

Description of class

CognacShow(_udf_)

The argument of constructor, **_udf_** is a name of UDF object to display. When using by GOURMET python command, it is set as the UDF object loaded in the window.

Description of Methods

- **all(***vtype,bc,color,drawrange***)** ··· Display of whole molecules

  – *vtype* ··· Drawing type
  'line'(default)/'ball-stick'/'rod'/'volume'

  – *bc* ··· Flag of whether to display molecules with applying boundary conditions.
  'off'(default)/'on'/'mol'/'atom'
  In the case of **on** or **mol**, the image molecules, whose center of mass is in a unit cell, are displayed. In the case of **atom**, the image atoms in a unit cell are displayed.

  – *color* ··· Option of display color
  'atom'(default)/'bond'/'molname'/'mol'
  When **atom** or **bond** are selected, the color of atoms and bonds is selected according to the atom type or the bond potential respectively.
  When **molname** is selected, the color of molecules is selected according to the name of molecules.
  When **mol** is selected, the molecules are drawn by the different color for every molecules.

  – *drawrange* ··· Option of the range of display
  The image molecules can be displayed in addition to those in basic unit cell. The range of the display is specified in the list as follows,
  [[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
  The number of cell to display in a,b and c axis is specified in *amin,amax,bmin...*, e.g. if the variable is set [[-1,1],[-1,1],[-1,1]], image molecules in the unit cells which are shifted by minus one and plus one for each axis is display in addition to the base (central) unit cell.

- **molecule(***target,vtype,bc,attrid,shift***)** ··· Display of arbitrary molecules

  – *target* ··· Specification of molecules
  Index of molecule(int), a name of molecules or a UDF location can be used to specify the molecules. When a name of molecules is specified, all molecules with this name are displayed.

  – *vtype* ··· Drawing type
  'line'(default)/'ball-stick'/'rod'/'volume'

  – *bc* ··· Flag of whether to display molecules with applying boundary conditions.
  'off'(default)/'on'/'mol'/'atom'

In the case of **on** or **mol**, the image molecules whose center of mass is in a unit cell are displayed.
In the case of **atom**, the images atoms in a unit cell are displayed.

– *attrid* ··· Option of display color
ID/'atom'(default)/'bond'
When **ID** is given, a color is selected from **ShapeDrawingAttr** file according to the **ID**.
Refer to "GOURMET Python script manual" for the details about **ShapeDrawingAttr** file.

– *shift* ··· Shift value for display
Shift value shifts the position of displaying molecules. The shift value is specified as a vector of interger,[a,b,c].

- **atom**(*target,vtype,bc,attrid*) ··· Display of arbitrary atoms

– *target* ··· Specification of atoms
A **list** of atom index, e.g. [**molIndex, atomIndex**] and [**atomIndex**], an atom type name or a UDF location can be used to specify the atoms. When the length of the **list** is 2 in the index list, the atom specified by a pair of **molIndex** and **atomIndex** is displayed.
When the length of the **list** is 1, atoms with **atomIndex** in all molecules are displayed.
When atom type name is given, all atoms of the atom type are displayed.
UDF location is effective only for single atom.

– *vtype* ··· Drawing type
'ball'(default)/'point'

– *bc* ··· Flag of whether to display atoms with applying boundary conditions.
'off'(default)/'on'/'atom'/'mol'
In the case of **on** or **atom**, the image atoms in a unit cell are displayed.
In the case of **mol**, the image atoms are selected so that the center of mass of image molecules including the atoms is in a unit cell.

– *attrid* ··· Option of display color **'atom'(default)/ID**
When **atom** is selected, the color of atoms is selected according to the atom type.
When **ID** is given, a color is selected from **ShapeDrawingAttr** file according to the **ID**.

- **bond**(*target,vtype,bc,attrid*) ··· Display of arbitrary bonds

– *target* ··· Specification of bonds
A **list** of bond index, e.g. [**molIndex, bondIndex**] and [**bondIndex**], a bond potential name or a UDF location can be used to specify the bonds. When the length of the **list** is 2 in the index list, the bond specified by a pair of **molIndex** and **bondIndex** is displayed.
When the length of the **list** is 1, bonds with **bondIndex** in all molecules are displayed.
When bond potential name is given, all bonds of the bond potential are displayed.
UDF location is effective only for single bond.

– *vtype* ··· Drawing type
'line'(default)/'rod'/'stick'

– *bc* ··· Flag of whether to display bond(s) with applying boundary conditions.
'off'(default)/'on'/'bond'/'mol'
In the case of **on** or **bond**, the image bonds in a unit cell are displayed.
In the case of **mol**, the image bonds are selected so that the center of mass of image molecules including the bonds is in a unit cell.

  - *attrid* $\cdots$ Option of display color **'atom'/'bond'(default)/ID**
    When **atom** or **bond** is selected, the color of bonds is selected according to the atom type or bond potential respectively.
    When **ID** is given, a color is selected from **ShapeDrawingAttr** file according to the **ID**.

- **volume(***index, attrid***)** $\cdots$ Display of arbitrary interaction sites

  The interaction sites defined by one atom or two atoms can be displayed. In the case of one atom sites, the sites are displayed by spheres, and in the case of two atom site, the sites are displayed by ellipsoids.

  - *index* $\cdots$ Specification of interaction site
    A **list** of interaction site, e.g. [**molIndex, siteIndex**] and [**siteIndex**] is used. When the length of the **list** is 2 in the index list, the bond specified by a pair of **molIndex** and **siteIndex** is displayed.
    When the length of the **list** is 1, bonds with **siteIndex** in all molecules are displayed.

  - *attrid* $\cdots$ **ID** of display color
    When **ID** is given, a color is selected from **ShapeDrawingAttr** file according to the **ID**.

- **cell(***color,drawrange***)** $\cdots$ Display of a unit cell

  A unit cell is displayed by line.

  - *color(int)* $\cdots$ Color is selected from **ShapeDrawingAttr** file according to the *color*. Default is **0** (white).

  - *drawrange* $\cdots$ Option of the range of display
    The image unit cell can be displayed in addition to basic unit cell. The range of the display is specified in the list as follows,
    [[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
    The number of cell to display in a,b and c axis is specified in *amin,amax,bmin...*, e.g. if the variable is set [[-1,1],[-1,1],[-1,1]], image unit cell which are shifted by minus one and plus one for each axis is display in addition to the base (central) unit cell.

- **boundary(***color,drawrange***)** $\cdots$ Display of boundary conditions

  The planes of the unit cell, where the boundary conditions are not applied, is displayed.

  - *color(int)* $\cdots$ Color is selected from **ShapeDrawingAttr** file according to the *color*. Default is **0** (white).

  - *drawrange* $\cdots$ Option of the range of display
    The boundary conditions for shifted unit cell can be displayed in addition to those of basic unit cell. The range of the display is specified in the list as follows,
    [[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
    The number of cell to display in a,b and c axis is specified in *amin,amax,bmin...*, e.g. if the variable is set [[-1,1],[-1,1],[-1,1]], image unit cell which are shifted by minus one and plus one for each axis is display in addition to the base (central) unit cell.

- **densityField(***griddensity,nmesh,atom_name, frame_attr,levelvalue,surf_color,clist***)** $\cdots$ Display of density distribution

Density profile obtained from the distribution of atoms is diplayed by color contour and isodensity surface.

- *griddensity* ··· Frag whether to use the grid density data in UDF. **'yes'/'no'(default)**
  If **'yes'** is selected, grid density data is necessary in UDF. If **'no'** is selected, density distribution is calculated from the coordinate of atoms in Structure data.

- *nmesh* ··· Number of grid point in one axis of unit cell
  If *griddensity* is set to **'yes'**, the definition of mesh in Grid density data is used, and this value is ignored. Default is **16**

- *atom_name* ··· Atom name to display the density
  List of Atom name is specified.

- *frame_attr* ··· Attribute of frame

- *levelvalue* ··· List of two value
  This is used to display isodensity surface. The averaged value of two specified values is used to define the isodensity surface.

- *surf_color* ··· Specification of the color of iso density surface. List of [R,G,B,transparency]. Default is **[1,1,0,1]**(yellow)

7.4.2 CognacBasicAnalysis.py

Description of class

CognacBasicAnalysis(*udffile,record***)**

Arguments:

- *udffile* ··· UDF file name to open
- *record* ··· Record no. to open

Description of methods

- **position(***atom***)** ··· Return of the position of atom
  - *atom* ··· **List** of atom index, **[molIndex, atomIndex]**

  return value ··· **Tuple** of the position, **(x, y, z)**

- **velocity(***atom***)** ··· Return of the velocity of atom
  - *atom* ··· **List** of atom index, **[molIndex, atomIndex]**

  return value ··· **Tuple** of the velocity, **(x, y, z)**

- **force(***atom***)** ··· Return of the force acting on atom
  - *atom* ··· **List** of atom index, **[molIndex, atomIndex]**

  return value ··· **Tuple** of the force, **(x, y, z)**

- **vector(***atom1,atom2***)** ··· Calculation of the differential vector between 2 atoms.
  - *atom1/atom2* ··· **List** of atom index. [**molIndex, atomIndex**]

  return value ··· **Tuple** of the differential vector, **(x2-x1, y2-y1, z2-z1)**

- **distance(***arg1,arg2***)** ··· Calculation of the distance between 2 atoms
  - Usage 1: Calculation of the length of all bonds specified by potential name
    - *arg1* ··· Potential name of bonds

    return value ··· **List** of bond lengths
  - Usage 2: Calculation of the length of the bonds specified by the bond index in specified molecules by the name of molecules.
    - *arg1* ··· Name of molecules
    - *arg2* ··· Bond index

    return value ··· **List** of bond lengths
  - Usage 3: Calculation of the length of the bond specified by the molecular index and bond index
    - *arg1* ··· **List** of index, [**molIndex, bondIndex**]

    return value ··· Bond length
  - Usage 4: Calculation of the distance between 2 atoms
    - *arg1/arg2* ··· **List** of atom index of two atoms, [**mol1, atom1**] and [**mol2, atom2**]

    return value ··· Distance between atoms

- **angle(***arg1,arg2,arg3***)**··· Calculation of the angle of three atoms
  - Usage 1: Calculation of all angles specified by angle potential name
    - *arg1* ··· Potential name of angles

    return value ··· **list** of angles
  - Usage 2: Calculation of the angles specified by the angle index in specified molecules by the name of molecules.
    - *arg1* ··· Name of molecules
    - *arg2* ··· Angle index

    return value ··· **list** of angles
  - Usage 3: Calculation of the angle specified by the molecular index and angle index
    - *arg1* ··· **list** of index, [**molIndex, angleIndex**]

    return value ··· Angle
  - Usage 4: Calculation of the angle of three atoms
    - *arg1/arg2/arg3* ··· **list**s of atom index of three atoms, [**mol1,atom1**],[**mol2,atom2**] and [**mol3,atom3**]

    return value ··· Angle

- **torsion(***arg1,arg2,arg3,arg4***)**$\cdots$ Calculation of the torsion angle of four atoms
  - Usage 1: Calculation of all torsion angles specified by torsion potential name
    - *arg1* $\cdots$ Potential name of torsions

    return value $\cdots$ **list** of torsion angles
  - Usage 2: Calculation of the torsions specified by the torsion index in specified molecules by the name of molecules.
    - *arg1* $\cdots$ Name of molecules
    - *arg2* $\cdots$ Torsion index

    return value $\cdots$ **list** of torsion angles
  - Usage 3: Calculation of the torsion specified by the molecular index and torsion index
    - *arg1* $\cdots$ **list** of index, [**molIndex, torsionIndex**]

    return value $\cdots$ Torsion angle
  - Usage 4: Calculation of the torsion of four atoms
    - *arg1/arg2/arg3/arg4* $\cdots$ **list**s of atom index of four atoms, [**mol1,atom1**],[**mol2,atom2**], [**mol3,atom3**] and [**mol4,atom4**]

    return value $\cdots$ Torsion angle

- **R2(***arg1***)**$\cdots$ Calculation of the square of end-to-end distance of molecule R^2
  - Usage 1: Calculation of R^2 of the specified molecule.
    - *arg1* $\cdots$ Molecular index

    return value $\cdots$ **Tuple** of R^2 and its components, $\boldsymbol{(R^2, (R_x^2, R_y^2, R_z^2))}$
  - Usage 2: Calculation of R^2 of all molecules specified by the name of molecules
    - *arg1* $\cdots$ Name of molecules

    return value $\cdots$ **list** of **tuple** consisting of R^2 and its components, $\boldsymbol{[(R^2, (R_x^2, R_y^2, R_z^2))...]}$

- **Rg2(***arg1***)**$\cdots$ Calculation of the square of radius of gyration of molecule Rg^2
  - Usage 1: Calculation of Rg^2 of the specified molecule.
    - *arg1* $\cdots$ Molecular index

    return value $\cdots$ **Tuple** of Rg^2 and its components, $\boldsymbol{(Rg^2, (Rg_x^2, Rg_y^2, Rg_z^2))}$
  - Usage 2: Calculation of Rg^2 of all molecules specified by the name of molecules
    - *arg1* $\cdots$ Name of molecules

    return value $\cdots$ **list** of **tuple** consisting of Rg^2 and its components, $\boldsymbol{[(Rg^2, (Rg_x^2, Rg_y^2, Rg_z^2))...]}$

- **centerofmass**(*molIndex, atomName*) $\cdots$ Calculation of center of mass of molecule or partial chain of moelcule
  - *molIndex* $\cdots$ Index of molecule
  - *atomName* $\cdots$ Name of atom or list of atom index for partial chain. If this argument is not specified, inertia axis of whole chain is calculated (default = None)

return value $\cdots$ Center of mass $\boldsymbol{r_{com}}$

- **system_centerofmass**() $\cdots$ Calculate the center of mass of the whole system.
  return value $\cdots$ Center of mass of the system $\boldsymbol{r_{com}}$

- **Xp**(*molIndex, p*)$\cdots$ Calculation of the normal coordinates of molecule.
  - *molIndex* $\cdots$ Molecular index
  - *p* $\cdots$ p-th mode to calculate. The default value is 1.

  return value $\cdots$ **Tuple** of the component of the normal mode, $\boldsymbol{(Xp_x, Xp_y, Xp_z)}$

- **inertiaAxis**(*molIndex*)$\cdots$ calculation of the inertia principal axis of molecule or part chain of molecule. A **list** of the inertia axis is sorted by the small order of moments **m**.
  - *molIndex* $\cdots$ Molecular index
  - *atomName* $\cdots$ Name of atom or list of atom index for partial chain. If this argument is not specified, inertia axis of whole chain is calculated (default = None)

  return value $\cdots$ **Tuple** of the inertia moments **m** and the inertia axes **v**,
  $\boldsymbol{([m1, m2, m3], [[v1_x, v1_y, v1_z], [v2_x, v2_y, v2_z], [v3_x, v3_y, v3_z]])}$

7.4.3 CognacGeometryAnalysis.py

Description of class

CognacGeometryAnalysis(*udffile,record*)

Arguments:

- *udffile* $\cdots$ UDF file name to open
- *record* $\cdots$ Record no. to open

Description of methods

- **gr**(*atomnames, width, max, type, minangle, maxangle*) $\cdots$ Calculation of pair-distribution functions
  - *atomnames* $\cdots$ Specification of a pair of atom name
    A **list** of atom name is specified. When the size of the **list** is 1, the pair distributions between the atoms of the same atom name is calculated, and when the size of the **list** is 2, the pair distributions between the atoms of two kind of atom name.
  - *width* $\cdots$ Width of bins of histogram
    The default value is 0.1.
  - *max* $\cdots$ Maximum distance of a histogram
    The default value is a half of the minimum cell length in the case of periodic boundary conditions.
  - *type* $\cdots$ Type of pair distribution **'all'/'inter'/'intra'**
    - **'all'** $\cdots$ Pair distribution of all pair specified in **Atom_Name**
    - **'inter'** $\cdots$ Pair distribution of pair between inter molecules.

    * **'intra'** ··· Pair distribution of pair between intra molecules.

  – *minangle* ··· Minimum angle of sector average

  – *maxangle* ··· Maximum angle of sector average

  return value ··· **list** of **list** consisting of distance and number density,
  [[**distance, number density, (number density of sector x,y,z)**]**...**]

- **gr_mol(***molnames, width, max, minangle, maxangle***)** ··· Calculation of pair-distribution functions of center of mass of molecules

  – *molnames* ··· Specification of a pair of molecular name
  A **list** of molecular name is specified. When the size of the **list** is 1, the pair distributions between the atoms of the same molecular name is calculated, and when the size of the **list** is 2, the pair distributions between the molecules of two kind of molecular name.

  – *width* ··· Width of bins of histogram
  The default value is 0.1.

  – *max* ··· Maximum distance of a histogram
  The default value is a half of the minimum cell length in the case of periodic boundary conditions.

  – *minangle* ··· Minimum angle of sector average

  – *maxangle* ··· Maximum angle of sector average

  return value ··· **list** of **list** consisting of distance and number density,
  [[**distance, number density**]**...**]

- **profile1D(***atomnames,direction,property,bin***)** ··· Calculation of the density distribution of position of atoms in 1D direction.

  – *atomnames* ··· Specification of atom names
  list of atom name is specified. Two or more atom names can be specified in the **list**.

  – *direction* ··· Axis **'X'/'Y'/'Z'(default)** to calculate the density distribution

  – *property* ··· Property **'density'(default)/'velocity'** to analyze

  – *bin* ··· Number of bins of histogram
  The default value is an integer part of the cell length.

  return value ··· **list** of **list** consisting of position and densities, [[**position,**[ϕ**1,** ϕ**2...**]]**...**]

- **profile1Dmol(***molnames,direction,property,bin***)** ··· Calculation of the density distribution of center of mass of molecules in 1D direction.

  – *molnames* ··· Specification of atom names
  list of mol name is specified. Two or more mol names can be specified in the **list**.

  – *direction* ··· Axis **'X'/'Y'/'Z'(default)** to calculate the density distribution

– *property* ··· Property **'density'(default)/'velocity'** to analyze

– *bin* ··· Number of bins of histogram
The default value is an integer part of the cell length.

return value ··· **list** of **list** consisting of position and densities, [[**position,**[ϕ**1,** ϕ**2...**]]**...**]

- **orientationOrderParameter(***molname,orientVec,refVec***)** ··· Calculation of orientation order parameters

The orientation order parameters of molecules are calculated.

– *molname* ··· Name of molecules to calculate the orientation order parameters

– *orientVec* ··· Specification of the vector to calculate orientation order parameters

A pair of atom index specifying vector **r** in **list**, e.g.[0,50], or the index of inertia axis,(0,1 or 2) is specified to calculate orientation order parameter. In *orientVec* is **None(default)**, the inertia principal axis(0) corresponding to minimum inertia moment is used.

– *refVec* ··· Specification of the referential vector

A vector $\mathbf{r_0}$ for the reference vector is specified by the **list**, **[x, y, z]**. In *refVec* is **None(default)**, the average value of the vector specified by *orientVec* $\langle \mathbf{r} \rangle$ is set as the referential vector.

return value ··· **list** of the orientation order parameter given by $(3\mathbf{u}\cdot\mathbf{u_0}^2-1)/2$
$\mathbf{u}, \mathbf{u_0}$:normalized vector of $\mathbf{r}, \mathbf{r_0}$ respectively. The length of the **list** is the number of molecules specified by the name of molecules.

- **orientationOrderParameterBond(***refVec***)** ··· Calculation of orientation bond order parameters

The orientation order parameters of all **Potential_Name** in **Set_of_Molecules** are calculated.

– *refVec* ··· Specification of the referential vector

A vector $\mathbf{r_0}$ for the reference vector is specified by the **list**, **[x, y, z]**. If *refVec* is **None(default)**, the average value of the vector of all bond of **Potential_Name**, $\langle \mathbf{r} \rangle$ is set as the referential vector.

return value ··· **list** of "bond potential name and **list** of the orientation order parameter given by $(3\mathbf{u}\cdot\mathbf{u_0}^2-1)/2$"
$\mathbf{u}, \mathbf{u_0}$:normalized vector of $\mathbf{r}, \mathbf{r_0}$ respectively. The length of the **list** is the number of molecules specified by the name of molecules.

- **normalizeOn()/normalizeOff()** ··· Toggle switch of whether to normalize the output of **gr** and **density**

In the case that the toggle is set on by **normalizeOn()**, the output is normalized by the number density of the total atoms in the system.

7.4.4 CognacTrajectoryAnalysis.py

Description of class

CognacTrajectoryAnalysis(*udffile***)**

Argument:

- *udffile* ··· UDF file name to open

Description of methods

- **molMsd(***molname, start_record, end_record, cancel_trans***)** ··· Calculation of the mean square displacements (MSD) of the center of mass of molecule.
  - *molname* ··· Name of molecules to calculate MSD
  - *start_record* ··· start record No. for sampling
  - *end_record* ··· end record No. for sampling
  - *cancel_trans* ··· if True, cancel the translation of the center of mass of the whole system

  return value ··· **list** of **tuple** consisting of time, MSD and its components,
  $[(\boldsymbol{Time}, \boldsymbol{MSD}, (\boldsymbol{MSD_x}, \boldsymbol{MSD_y}, \boldsymbol{MSD_z}))\boldsymbol{...}]$

- **atomMsd(***molname, atomIndex, start_record, end_record, cancel_trans***)** ··· Calculation of the mean square displacements (MSD) of specified atoms.

  The MSD of the atoms in the molecules specified by atom index and the name of molecules are calculated.
  - *molname* ··· Name of molecules
  - *atomIndex* ··· **list** of atom index, e.g.**[0,1,2,3,4]**
  - *start_record* ··· start record No. for sampling
  - *end_record* ··· end record No. for sampling
  - *cancel_trans* ··· if True, cancel the translation of the center of mass of the whole system

  return value ··· **list** of **tuple** consisting of time, MSD and its components,
  $[(\boldsymbol{Time}, \boldsymbol{MSD}, (\boldsymbol{MSD_x}, \boldsymbol{MSD_y}, \boldsymbol{MSD_z}))\boldsymbol{...}]$

- **normalCoodinate(***molname,p,start_record,end_record***)** ··· Calculation of the self-correlation function of normal coordinates, *Cp*.

  The average value of *Cp* of the molecules specified by the name of molecules is calculated.
  - *molname* ··· Name of molecules
  - *p* ··· p-th mode to calculate. The default value is 1.
  - *start_record* ··· start record No. for sampling
  - *end_record* ··· end record No. for sampling

  return value ··· **list** of **tuple** consisting of time, *Cp* and its components,
  $[(\boldsymbol{Time}, \boldsymbol{Cp}, (\boldsymbol{Cp_x}, \boldsymbol{Cp_y}, \boldsymbol{Cp_z}))\boldsymbol{...}]$

- **vectorAutoCorrelation(***molname,atomIndex,start_record,end_record***)** ··· Calculation of the self-correlation function of the specified vector between atoms.

  - *molname* ··· Name of molecules
  - *atomIndex* ··· A pair of atom index in **list**. e.g.[**0,10**]
  - *start_record* ··· start record No. for sampling
  - *end_record* ··· end record No. for sampling

  return value ··· **list** of **tuple** consisting of time, self-correlation and its components,
  $[(\boldsymbol{Time}, \boldsymbol{Cvec}, (\boldsymbol{Cvec_x}, \boldsymbol{Cvec_y}, \boldsymbol{Cvec_z}))\boldsymbol{...}]$

- **atomVelocityAutoCorrelation(***atom_name,start_record,end_record***)** ··· Calculation of the velocity self-correlation function of the specified atom name.

  - *atom_name* ··· Name of atom
  - *start_record* ··· start record No. for sampling
  - *end_record* ··· end record No. for sampling

  return value ··· **list** of **tuple** consisting of time, self-correlation and its components,
  $[(\boldsymbol{Time}, \boldsymbol{Cvel}, (\boldsymbol{Cvel_x}, \boldsymbol{Cvel_y}, \boldsymbol{Cvel_z}))\boldsymbol{...}]$

- **molVelocityAutoCorrelation(***mol_name,start_record,end_record***)** ··· Calculation of the velocity self-correlation function of the center of mass of specified mol name.

  - *mol_name* ··· Name of molecule
  - *start_record* ··· start record No. for sampling
  - *end_record* ··· end record No. for sampling

  return value ··· **list** of **tuple** consisting of time, self-correlation and its components,
  $[(\boldsymbol{Time}, \boldsymbol{Cvel}, (\boldsymbol{Cvel_x}, \boldsymbol{Cvel_y}, \boldsymbol{Cvel_z}))\boldsymbol{...}]$

- **forceAutoCorrelation(***atom_name,start_record,end_record***)** ··· Calculation of the self-correlation function of the force acting atoms with specified atom name.

  - *atom_name* ··· Name of atom
  - *start_record* ··· start record No. for sampling
  - *end_record* ··· end record No. for sampling

  return value ··· **list** of **tuple** consisting of time, self-correlation and its components,
  $[(\boldsymbol{Time}, \boldsymbol{Cforce}, (\boldsymbol{Cforce_x}, \boldsymbol{Cforce_y}, \boldsymbol{Cforce_z}))\boldsymbol{...}]$

- **stressAutoCorrelation(***atom_name,start_record,end_record***)** ··· Calculation of the self-correlation function of stress of the system.

  - *start_record* ··· start record No. for sampling
  - *end_record* ··· end record No. for sampling

  return value ··· **list** of **tuple** consisting of time, self-correlation and its components,
  $[(\boldsymbol{Time}, \boldsymbol{Cpress}, (\boldsymbol{Cvel_x x}, \boldsymbol{Cvel_y y}, \boldsymbol{Cvel_z z}), (\boldsymbol{Cvel_x y}, \boldsymbol{Cvel_y z}, \boldsymbol{Cvel_z x}))\boldsymbol{...}]$

7.4.5 CognacFileConvert.py

Description of class

CognacFileConvert(*udffile,record***)**

Arguments:

- *udffile* ··· UDF file name to open
- *record* ··· Record no. to open

Description of methods

- **writePDB(***file,scale***)** ··· Conversion of UDF data into Protein Data Bank(PDB) format file.
  - *file* ··· Output file name
  - *scale* ··· Scale factor

    When converting into a real unit system, the position data in the UDF is converted to the real unit by multiplying by the scale factor. For example, when **COGNAC** is calculated using a model which has unit length 1σ=4.0Å, then scale factor is set to 4.0, the position will be converted for output with the length of Å unit. The default value is 1.0.

- **writeCAR(***file,scale***)** ··· Conversion of UDF data into Accelrys **car** format file.
  - *file* ··· Output file name
  - *scale* ··· Scale factor

    When converting into a real unit system, the position data in the UDF is converted to the real unit by multiplying by the scale factor. The default value is 1.0.

- **writeXYZ(***file,scale***)** ··· Conversion of UDF data into **XYZ** format file.
  - *file* ··· Output file name
  - *scale* ··· Scale factor

    When converting into a real unit system, the position data in the UDF is converted to the real unit by multiplying by the scale factor. The default value is 1.0.

- **writeAVS(***file***)** ··· Conversion of **Grid_Density** data in UDF into AVS regular mesh & field format file.
  - *file* ··· Output file name

- **write(***type,file,scale***)** ··· Conversion of UDF data into specified format file.
  - *type* ··· Output file format (**'pdb'/'car'/'XYZ'/'AVS'**)
  - *file* ··· Output file name
  - *scale* ··· Scale factor

    When converting into a real unit system, the position data in the UDF is converted to the real unit by multiplying by the scale factor. The default value is 1.0. If **'AVS'** is selected in the *type*, this parameter is not used.

7.4.6 CognacUtility.dll/so

This is a library used by the script for the conversion of coordinates by boundary conditions, the calculation of self-correlation functions, etc. The functions in the library is used as a function of Python, and input and output data are independent from UDF.

Description of functions

- Functions for the operation of boundary conditions
  - **setCell(***arg1***)** $\cdots$ Registration of unit cell data
    - *arg1* $\cdots$ **Tuple** of the unit cell data, **(a,b,c,alpha,beta,gamma)**
      Angles should be given by degree.

    return value $\cdots$ None
  - **getCell()** $\cdots$ Return of the registered unit cell data

    return value $\cdots$ **Tuple** of the unit cell data
  - **getH()** $\cdots$ Return of the conversion matrix **H** calculated from the registered unit cell data

    $\mathbf{r} = \mathbf{H}\mathbf{r}', \mathbf{r}$: real coordinate, $\mathbf{r}'$: normalized coordinate

    return value $\cdots$ **H**
  - **getHinv()** $\cdots$ Return of the conversion matrix $\mathbf{H}^{-1}$ calculated from the registered unit cell data

    $\mathbf{r}' = \mathbf{H}^{-1}\mathbf{r}$

    return value $\cdots$ $\mathbf{H}^{-1}$

    H and $\mathbf{H}^{-1}$ is given in a form of **(xx, yy, zz, yz, xz, xy)** with considering symmetry.
  - **getVolume()** $\cdots$ Return of the unit cell volume

    return value $\cdots$ Volume
  - **setBoundary(***arg1***)** $\cdots$ Registration of boundary conditions
    - *arg1* $\cdots$ **Tuple** of boundary conditions, **(X,Y,Z)**
      X, **Y** and **Z** are integers which specify boundary conditions.
      0(None)/1(Periodic)/2(Reflective1)/3(Reflective2)

    return value $\cdots$ None
  - **getBoundary()** $\cdots$ Return of the registered boundary conditions

    return value $\cdots$ **Tuple** of boundary conditions, **(X,Y,Z)**
  - **distanceWithBoundary(***arg1,arg2***)** $\cdots$ Calculation of the differential vector of minimum image
    - *arg1/arg2* $\cdots$ **Tuple** of two positions, **(x1,y1,z1)** and **(x2,y2,z2)**

    return value $\cdots$ **Tuple** of the differential vector, **(dx,dy,dz)** of minimum image

– **positionWithBoundary(***arg1***)** ··· Calculation of the position of the image in the unit cell, $0 \leq x, y$ and $z < Cell_{Max}$.

∗ *arg1* ··· **Tuple** of the position, **(x,y,z)** to calculate the image position

return value ··· **Tuple** of the position in a unit cell, **(x',y',z')**

– **getShift(***arg1***)** ··· Calculation of the number of shifts of the cell size to convert the given position into the position in a unit cell.

∗ *arg1* ··· **Tuple** of the position, **(x,y,z)** to calculate

return value ··· **Tuple** of the shift value, **(shift X, shift Y, shift Z)**
The position in a unit cell **r'** are given by $\mathbf{r}' = \mathbf{H}(\mathbf{H}^{-1}\mathbf{r} - \mathbf{shift})$

– **positionWithShift(***arg1,arg2***)** ··· Calculation of the image position shifted by a given shift vector.

∗ *arg1* ··· **Tuple** of the position, **(x,y,z)** to calculate
∗ *arg2* ··· **Tuple** of the shift vector, **(shift x,shift y,shift z)**

return value ··· **Tuple** of the shifted position, **(x',y',z')**

• Functions for the storage of sequences of data

In order to analyze sequences of data as described later, it is necessary to store the data beforehand. Inside the library, these data are stored in the form of **map**⟨**string, vector**⟨**double**⟩ ⟩ or **map**⟨**string, vector**⟨**Vector3d**⟩ ⟩ (**Vector3d** is a **class** which has **double x, y** and **z** in members). The following methods are offered as a Python interface to operate these **map**s.

– **pushScalar(***arg1,arg2***)** ··· Store of a scalar data

This function executes **push_back** the scalar data to the **vector** specified by **key** internally.

∗ *arg1* ··· Key(string)
∗ *arg2* ··· Scalar data

return value ··· None

– **pushVector(***arg1,arg2***)** ··· Store of a 3D vector data

This function executes **push_back** the 3D vector data to **vector** specified by **key** internally.

∗ *arg1* ··· Key(string)
∗ *arg2* ··· **Tuple** of the 3D vector data, **(x,y,z)**

return value ··· None

– **sizeScalar(***arg1***)** ··· Return the size of **vector**⟨**double**⟩ specified by **key**

∗ *arg1* ··· Key(string)

return value ··· Size of **vector**

– **sizeVector(***arg1***)** ··· Return the size of **vector**⟨**Vector3d**⟩ specified by **key**

∗ *arg1* ··· Key(string)

return value ··· Size of **vector**

– **clearScalarMap()** ··· Clear of the data in all scalar **map**

return value ··· None

– **clearVectorMap()** ··· Clear of the data in all **Vector3d map**

return value ··· None

- Functions for the analysis of the sequences of data
  - **pairDistribution(***arg1,arg2,arg3,arg4,arg5,arg6***)** ··· Calculation of pair distribution functions
    - *arg1* ··· **Tuple** of **list** consisting of **key** of **Vector3d map**, ([**key1**] or [**key1, key2**])
      When one key is given, the pair distribution between the data specified by **key1** and **key1** is calculated, and when two keys are given, the pair distribution between the data specified by **key1** and **key2** is calculated.
    - *arg2* ··· Number of bins of histogram
    - *arg3* ··· Width of each bin of histogram
    - *arg4* ··· Calculate pair distribution of intra molecules 1, inter molecules -1 and all pair 0
    - *arg5* ··· Minimum angle of sector average
    - *arg6* ··· Maximum anfle of sector average

    return value ··· **list** of probability, [**gr,gr_x,gr_y,gr_z**]

    The probability is a number of data specified by **key1** or **key2** which exists in the distance range of section i, $(i * width < r < (i+1) * width)$, per one data specified by **key1**.
    gr_x,gr_y,gr_z contain pair distribution which is in the sector between minimum and maximum angle from each axis
  - **msd(***arg1***)** ··· Calculation of mean square displacements, MSD.

$$MSD(|i-j|) =< (\mathbf{V}(i) - \mathbf{V}(j))^2 > \tag{7.1}$$

    V(i):i-th data in **Vector3d map**
    - *arg1* ··· **Key** of **Vector3d map**
      When the *arg1* is not given, MSD from all vector data stored in the **map** is calculated.

    return value ··· **list** of the components of the MSD, [[**MSD_x,MSD_y,MSD_z**]**...**]

    The index of outer **list** corresponds to $|i-j|$ in eq.7.1.

  - **scalarCorrelation(***arg1,arg2***)** ··· Calculation of correlation functions C_s of a sequence of scalar data

$$C_s(|i-j|) =< S(i) * S(j) > \tag{7.2}$$

    S(i):i-th data in scalar **map**
    - Usage 1: No argument
      The auto correlation of all the sequences of scalar **map** is calculated.
    - Usage 2: One argument, *arg1*
      The auto correlation of a sequence of scalar data stored with the key *arg1* is calculated.
    - Usage 3: Two arguments, *arg1* and *arg2*
      The cross correlation of two sequences of scalar data stored with the keys *arg1* and *arg2* is calculated.

    return value ··· **list** of $C_s(|i-j|)$

    The index of the **list** corresponds to $|i-j|$.

  - **vectorCorrelation(***arg1,arg2***)** ··· Calculation of correlation functions C_v of a sequence of 3D vector data

$$C_v(|i-j|) =< \mathbf{V}(i) * \mathbf{V}(j) > \tag{7.3}$$

    V(i):i-th Vector data in vector3d map

* Usage 1: No argument
  The auto correlation of all the sequences of **Vector3d map** is calculated.
* Usage 2: One argument, *arg1*
  The auto correlation of a sequence of vector data stored with the key *arg1* is calculated.
* Usage 3: Two arguments, *arg1* and *arg2*
  The cross correlation of two sequences of vector data stored with the keys *arg1* and *arg2* is calculated.

return value $\cdots$ **list** of $C_v(|i-j|)$, **[[corr_x,corr_y,corr_z]...]**

The index of the **list** corresponds to $|i-j|$.

Examples of how to use

Simple examples to use the functions of **CognacUtility** are shown.
Refer to the "CognacGeometryAnalysis.py"/"CognacTrajectoryAnalysis.py" for more examples.

1. Example to use functions related to boundary conditions

```
Python 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> from CognacUtility import *          # import module
>>> cell = (10,10,10,90,90,90)           # Setup of unit cell
>>> setCell(cell)
>>> boundary = (1,1,1)                   # Setup of boundary conditions
>>> setBoundary(boundary)                # Set all cell to periodic boundary
>>> print getVolume()                    # Calculate volume of unit cell
1000.0
>>> pos1 = (5,15,-5)
>>> pos2 = (7,13,-9)
>>> print positionWithBoundary(pos1)     # Calculate the potision of
(5.0, 5.0, 5.0)                          # image of pos1 in unit cell
>>> print positionWithBoundary(pos2)
(7.0, 3.0, 1.0)
>>> print distanceWithBoundary(pos1,pos2)# Calulate the vector of minimum
(2.0, -2.0, -4.0)                        # image between pos1 and pos2
>>> shift = getShift(pos1)               # Calculate the shift value of pos1
>>> print shift                          #
(0.0, 1.0, -1.0)
>>> print positionWithShift(pos2,shift)  # Move pos2 with shift
(7.0, 3.0, 1.0)
>>>
```

2. Example to use the functions related to the analysis of the sequences of data

```
Python 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> from CognacUtility import *           # import module
>>> from math import *                    # In order to use Pi, import math
                                          #  module
>>> for i in range(0,10):                 # Set scalar data sequence which have
    pushScalar("A",i)                     #  keys "A" and "B"
    pushScalar("B",i*i)                   # map["A"]=[0,1,2,3...]
                                          # map["B"]=[0,1,4,9...]
```

```
>>> corrAA = scalarCorrelation("A")       # Calculate the auto-correlation
                                          # function of map["A"]
>>> corrAB = scalarCorrelation("A","B")   # Calculate the cross-correlation
                                          # function between map["A"] and map["B"]
>>> corrAll = scalarCorrelation()         # Calculate averaged auto-correlation
                                          # function of all scalar map
>>> for i in range(0,len(corrAA)):        # Print the result in order, |i-j|=0,1,2...
    print corrAA[i],corrAB[i],corrAll[i]


28.5 202.5 780.9
26.6666666667 180.0 656.0
24.5 157.5 535.5
22.0 135.0 421.0
19.1666666667 112.5 314.5
16.0 90.0 218.4
12.5 67.5 135.5
8.66666666667 45.0 69.0
4.5 22.5 22.5
0.0 0.0 0.0
>>>


>>> for i in range (0,10):                # Set data of cubic lattice
            for j in range(0,10):         # with the key "pos"
                for k in range(0,10):
                        pushVector("pos",(i,j,k))


>>> setCell( (10,10,10,90,90,90) )        # Setup of unit cell
>>> setBoundary( (1,1,1) )                # and bounadry conditions
>>>
>>> bin = 50
>>> width = 0.1
>>> pr = pairDistribution( tuple(["pos"]), bin, width) # Calculate
>>>                                       # pair distribution function
>>> for i in range (0,bin):               # Calculate pair distribution by
         r = (i+0.5)*width                # deviding surface area of sphere
         gr = pr[i]/(4*pi*r*r*width)
         print (i+0.5)*width, gr

0.05 0.0
0.15 0.0
0.25 0.0
0.35 0.0
0.45 0.0
0.55 0.0
0.65 0.0
0.75 0.0
0.85 0.0
0.95 3.17745887234
1.05 1.73403273305
1.15 0.0
1.25 0.0
1.35 0.0
1.45 4.54642351532
1.55 0.0
```

```
1.65 0.0
1.75 2.08083927966
1.85 0.0
1.95 0.628458608873
2.05 0.568641013739
2.15 0.0
2.25 3.77633795198
...
(skip)

>>>
>>> clearVectorMap()                    # Clear vector map
>>> for i in range(0,10):               #
    pushVector( "pos1", (i,i*2,i*3) )   # Set arbitrary value to
    pushVector( "pos2", (i,i/2,i/3) )   # map["pos1"] and map["pos2"]

>>> gpos1 = msd("pos1")                 # Calculate MSD of pos1
>>> gpos2 = msd("pos2")                 # Calculate MSD of pos2
>>> gave = msd()                        # Calculate averaged MSD of
                                        # pos1 and pos2
>>> for i in range(0,len(g1ave)):       # Print the results
    print gave[i],gpos1[i],gpos2[i]


[0.0, 0.0, 0.0] [0.0, 0.0, 0.0] [0.0, 0.0, 0.0]
[1.0, 2.22222222222, 4.66666666667] [1.0, 4.0, 9.0] \\
[1.0, 0.444444444444, 0.333333333333]
[4.0, 8.5, 18.3125] [4.0, 16.0, 36.0] [4.0, 1.0, 0.625]
[9.0, 19.1428571429, 41.0] [9.0, 36.0, 81.0] [9.0, 2.28571428571, 1.0]
[16.0, 34.0, 73.0] [16.0, 64.0, 144.0] [16.0, 4.0, 2.0]
[25.0, 53.0, 113.9] [25.0, 100.0, 225.0] [25.0, 6.0, 2.8]
[36.0, 76.5, 164.0] [36.0, 144.0, 324.0] [36.0, 9.0, 4.0]
[49.0, 103.666666667, 223.333333333] [49.0, 196.0, 441.0] \\
[49.0, 11.3333333333, 5.66666666667]
[64.0, 136.0, 291.25] [64.0, 256.0, 576.0] [64.0, 16.0, 6.5]
[81.0, 170.0, 369.0] [81.0, 324.0, 729.0] [81.0, 16.0, 9.0]
>>>
```

7.5 Analysis using Action

COGNAC has a function of the analysis using **Action** of GOURMET, for easy use of Python scripts described above. Main commands can be selected from the command list as it is shown in Figure 7.1 by clicking the root icon showing the UDF file name, with the right button of mouse.

The almost of all commands and parameters used by **Action** correspond to those of functions of Python scripts. The action commands for **COGNAC** UDF data are listed below.

- **ANALYSIS_1D_profile...**

  Description : Calculation and plotting of 1D profile.

  The command is selectable at the root icon of tree structure.
  The arguments shown in Figure 7.2 are explained below.

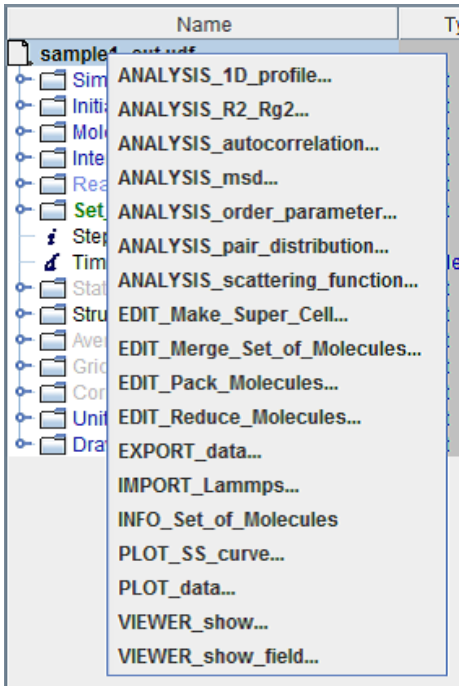

Figure 7.1: Command list displayed by **Action**

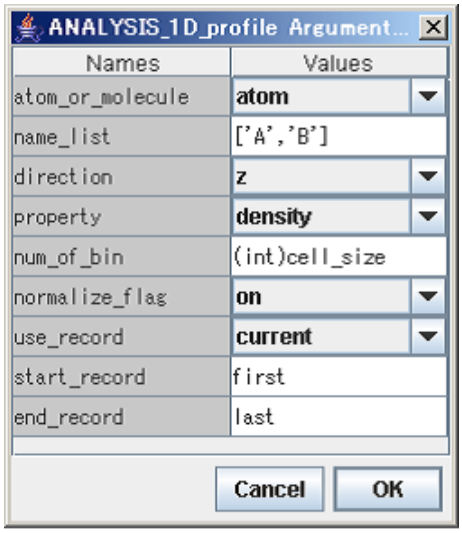

Figure 7.2: Arguments of **ANALYSIS_1D_profile...**

- **atom_or_molecule** ··· It is selected from **atom/molecule**
- **name_list** ··· List of atom or mol name, e.g. ['A','B'], to calculate the density profile.
- **direction** ··· Direction of axis
  It is selected from **z/y/x**.
- **property** ··· Property to analyze
  It is selected from **density/velocity**
- **num_of_bin** ··· Number of bin (integer)
  The default will set a number of bin as the integer part of unit cell length of specified axis.
- **normalize_flag** ··· Flag of normalization
  It is selected from **on/off**.

- **use_record** ··· Selection of target record fot analysis. It is selected from **current/range**. In the case when **current** is selected, the only current record is used for analysis, and when **range** is selected, the average over the range of records is calculated.
- **start_record** ··· In the case wehn **range** is selected in **use_record**, the initial record no. is specified.
- **end_record** ··· In the case wehn **range** is selected in **use_record**, the final record no. is specified.

- **ANALYSIS_R2_Rg2...**

Description : Calculation of the end-to-end distance R^2 and radius of gyration Rg^2 of specified molecules

This command is selectable at the root icon of tree structure.
The arguments shown in Figure 7.3 are explained below.

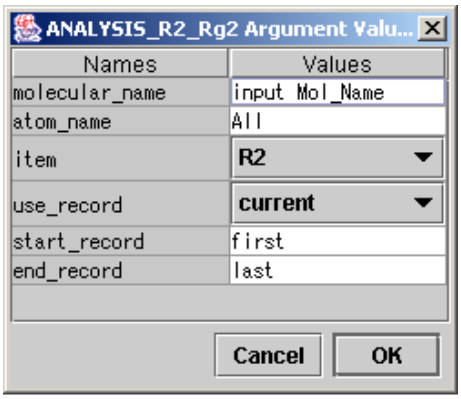

Figure 7.3: Arguments of **ANALYSIS_R2_Rg2...**

- **molecular_name** ··· Name of molecules to calculate R^2 and Rg^2
  The default value will select the name of molecules of the first molecule in a **Set_of_Molecules** (**Set_of_Molecules.molecule[0]**).
- **atom_name** ··· Name of atom to calculate R^2 and Rg^2
  This parameter is effective to calculate the shape of partical chain in block copolymers.
  The parameter **'all'** calculate the R^2 and Rg^2 of complete chain.

- **item** ··· Item to calculate
  It is selected from **R2/Rg2/both**
- **use_record** ··· Selection of target record fot analysis. It is selected from **current/range**. In the case when **current** is selected, the only current record is used for analysis, and when **range** is selected, the analysis is executed for specified range of records.
- **start_record** ··· In the case wehn **range** is selected in **use_record**, the initial record no. is specified.
- **end_record** ··· In the case wehn **range** is selected in **use_record**, the final record no. is specified.

In the case when **current** is selected in **use_record**, the list of R^2 and/or Rg^2 of molecules specified by the name will be printed in a Python log window.
In the case when **range** is selected in **use_record**, the average of the list of R^2 and/or Rg^2 of specified molecules in each record will be printed in a Python log window.

- **ANALYSIS_autocorrelation...**

  Description : Calculation and plotting of the autocorrelation function

  This command is selectable at the root icon of tree structure.
  The arguments shown in Figure 7.4 are explained below.

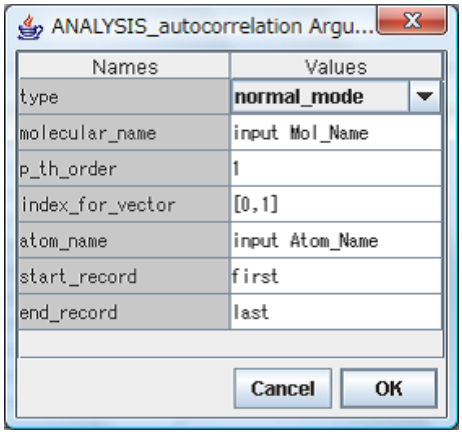

Figure 7.4: Arguments of **ANALYSIS_autocorrelation...**

  - **type** ··· Type of property to calculate autocorrelation
    It is selected from **normal_mode/vector/atom_velocity/mol_velocity/force/stress**.
  - **molecular_name** ··· Name of molecules to calculate autocorrelation
    The default value will select the name of molecules of the first molecule in a **Set_of_Molecules** (**Set_of_Molecules.molecule[0]**). In the case of **type** is **stress**, this paremeter is not used.
  - **p_th_order** ··· p-th order of Normal mode
    This argument is effective only when **normal_mode** is selected in **type**.
  - **index_for_vector** ··· A pair of atom index to define the vector to calculate autocorrelation.
    This argument is effective only when **vector** is selected in **type**.
  - **atom_name** ··· Atom name to calculate velocity of force autocorrelation function.
    This argument if effective only when **atom_velocity** or **force** is selected in **type**.
  - **start_record** ··· Start record No. for sampling. Default value (**first**) corresponds **Record 0**
  - **end_record** ··· End record No. for sampling. Default value (**last**) corredponds the last record.
    In the case that both **start_record** and **end_record** take default value, Only records which has a data **Time** will be taken for sampling, i.e. such a record which has a name "Initialize" will be skipped.

- **ANALYSIS_msd...**

  Description : Calculation and plotting of the mean square displacements, MSD

  This command is selectable at the root icon of tree structure.
  The arguments shown in Figure 7.5 are explained below.

  - **type** ··· Item to calculate MSD
    It is selected from **molecule/atom**.

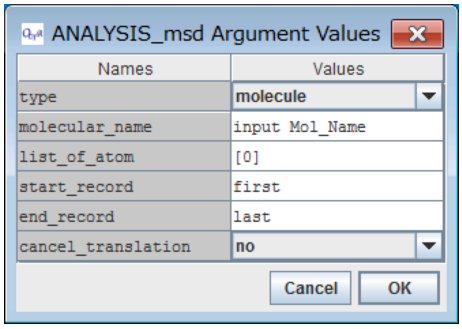

Figure 7.5: Arguments of **ANALYSIS_msd...**

- **molecular_name** ··· Name of molecules to calculate MSD
  The default value will select the name of molecules of the first molecule in a **Set_of_Molecules** (**Set_of_Molecules.molecule[0]**).
- **list_of_atom** ··· List of atom index to calculate MSD
  This argument is effective only when **atom** is selected in **type**.
- **start_record** ··· Start record No. for sampling. Default value (**first**) corresponds **Record 0**
- **end_record** ··· End record No. for sampling. Default value (**last**) corredponds the last record.
  In the case that both **start_record** and **end_record** take default value, Only records which has a data **Time** will be taken for sampling, i.e. such a record which has a name "Initialize" will be skipped.
- **cancel_translation** ··· If yes, cancel the translation of the center of mass of the whole system.

- **ANALYSIS_order_parameter...**

Description : Calculation of the orientation order parameter.

This command is selectable at the root icon of tree structure.
The arguments shown in Figure 7.6 are explained below.

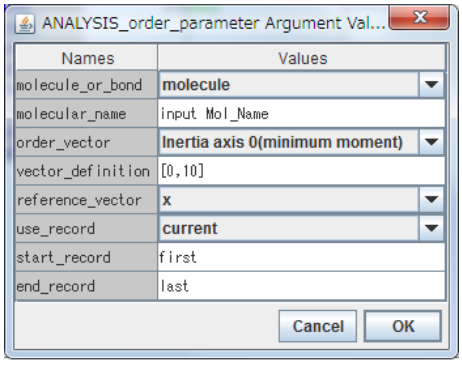

Figure 7.6: Arguments of **ANALYSIS_order_parameter...**

- **molecule_or_bond** ··· Selection of molecular order or bond order
  In the case when **molecules** is selected, molecular name and molecular vector are specified in **molecular_name**, **order_vector** and **vector_definition**. In the case when **bond** is selected, these input values are not used, and order paramters of all bond of **Potential_Name** used in **Set_of_Molecules** are calculated.

- **molecular_name** ··· Name of molecules to calculate orientation order parameter
  The default value will select the name of molecules of the first molecule in a **Set_of_Molecules** (**Set_of_Molecules.molecule[0]**).
- **order_vector** ··· Type of vector to calculate order parameter
  It is selected from **Inertia axis 0(minimum moment)/ Inertia axis 1/ Inertia axis 2(maximum moment)/ Vector between atoms**.
- **vector_definition** ··· A pair of atom index to define the vector to calculate order parameter
  This argument is effective only when **Vector between atoms** is selected in **order_vector**.
- **reference_vector** ··· Referential vector
  It is selected from **x/y/z/average**. **x**,**y** and **z** specify the axis for the referential vector and **average** specifies the averaged vector of **order_vector** over molecules or all bond vector for the referential vector.
- **use_record** ··· Selection of target record fot analysis. It is selected from **current/range**. In the case when **current** is selected, the only current record is used for analysis, and when **range** is selected, the analysis is executed for specified range of records.
- **start_record** ··· In the case wehn **range** is selected in **use_record**, the initial record no. is specified.
- **end_record** ··· In the case wehn **range** is selected in **use_record**, the final record no. is specified.

In the case when **molecule** is selected in **molecule_or_bond**, and if **current** is selected in **use_record**, the list of order parameters of molecules specified by the name will be printed in a Python log window. If **range** is selected in **use_record**, the average of the order parameters of specified molecules in each record will be printed in a Python log window and the results are plotted.
In the case when **bond** is selected in **molecule_or_bond**, and if **current** is selected in **use_record**, the list of order parameters of all bond will be printed for all **Potential_Name** used in **Set_of_Molecules**. If **range** is selected in **use_record**, the average of the order parameters of all bond in each record will be printed by every **Potential_Name** in a Python log window and the results are plotted.

- **ANALYSIS_pair_distribution...**

Description : Calculation and plotting of the pair distribution function

This command is selectable at the root icon of tree structure.
The arguments shown in Figure 7.7 are explained below.

- **atom_or_molecule** ··· Selection of Atom or Molecule. When **molecule** is selected, pair distribution function of the center of mass of molecules is calculated.
- **name_list** ··· List of atom or molecular name, e.g. **['A','B']**, to calculate the pair distribution function
- **width_of_bin** ··· Width of bin
- **max_range** ··· Maximum distance
  The default value will set a half of minimum cell length to **max_range**.
- **type** ··· Type of pair distribution **'all'/'inter'/'intra'**
  - **'all'** ··· Pair distribution of all pair specified in **name_list**
  - **'inter'** ··· Pair distribution of pair between inter molecules.
  - **'intra'** ··· Pair distribution of pair between intra molecules.
- **normalize_flag** ··· Flag for normalization
  It is selected from **on/off**.
- **plot_sector_average** ··· Flag for calculating sector average along x,y, and z axis. It is selected from **on/off**.

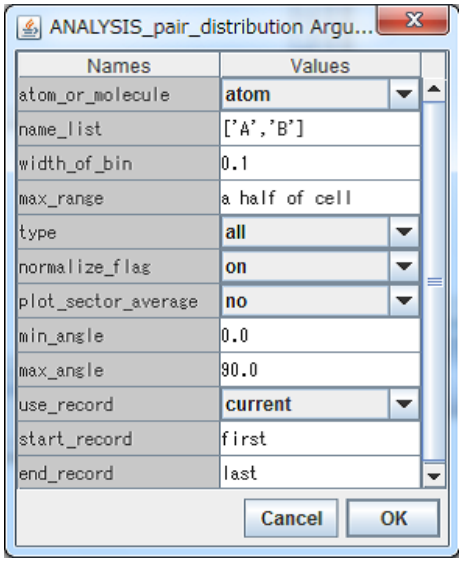

Figure 7.7: Arguments of **ANALYSIS_pair_distribution...**

- **min_average** ··· Minimum angle for sector average
- **max_average** ··· Maximum angle for sector average
- **use_record** ··· Selection of target record fot analysis. It is selected from **current/range**. In the case when **current** is selected, the only current record is used for analysis, and when **range** is selected, the average over the range of records is calculated.
- **start_record** ··· In the case wehn **range** is selected in **use_record**, the initial record no. is specified.
- **end_record** ··· In the case wehn **range** is selected in **use_record**, the final record no. is specified.

- **ANALYSIS_scattering_function...**

Description : Calculation and plotting of scattering function from denisty data on the grid point.

This command is selectable at the root icon of tree structure.
The arguments shown in Figure 7.8 are explained below.

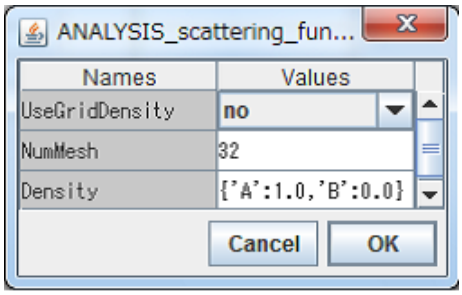

Figure 7.8: Arguments of **ANALYSIS_scattering_function...**

- **UseGridDensity** ··· Flag whether to use **Grid_Denisty** in the output of **COGNAC**, or calculate the density distribution in the command. It is selected from **on/off**.
- **NumMesh** ··· Number of mesh, when **UseGridDensity** is 'no'. If an integer n is given, all axis are devided by the same number, while a list *[nx,ny,nz]* is given, each axis is devied by the specified number. When **UseGridDensity** is 'yes', this argument is ignored.

- **Density** ··· Coefficients of each **Atom_Name** to density, e.g. when the argument is specified as **{'A':1.0,'B':0.0}**, density is given based on that of **Atom_Name** 'A'.
- **use_record** ··· Selection of target record fot analysis. It is selected from **current/range**. In the case when **current** is selected, the only current record is used for analysis, and when **range** is selected, the average over the range of records is calculated.
- **start_record** ··· In the case wehn **range** is selected in **use_record**, the initial record no. is specified.
- **end_record** ··· In the case wehn **range** is selected in **use_record**, the final record no. is specified.

- **EXPORT_data...**

Description : Conversion of the UDF to other file format. The data in current record will be output to new file.

This command is selectable at the root icon of tree structure.
The arguments shown in Figure 7.9 are explained below.

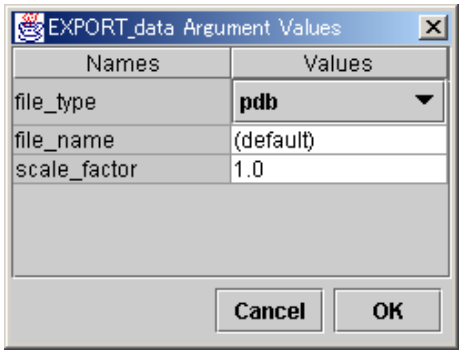

Figure 7.9: Arguments of **EXPORT_data...**

- **file_type** ··· Format to convert
  It is selected from **udf/pdb/car/XYZ/LAMMPS/AVS**. The description of each format is the follows:
  - **udf** ··· **udf** is selected, new udf file which includes only current record will be output without data conversion.
  - **pdb** ··· **Strcuture** data is coverted to Proten Data Bank format. The CONECT information is also created.
  - **car** ··· **Structure** data is converted to Accelrys car file format.
  - **XYZ** ··· **Strcuture** data is converted to XYZ format.
  - **LAMMPS** ··· The information of simulation system. e.g. number of atoms, unit cell size, the coordinate of atoms and the information of connction, is converted to Molecular dynamcis package "LAMMPS" data format. See the details of "LAMMPS" and its data format on the web page (http://lammps.sandia.gov/).
  - **AVS** ··· **Grid_Density** is converted to AVS structure mesh format.
- **file_name** ··· File name to output
  The default value will set the UDF file name to the output file name with replacing the extension.
- **scale_factor** ··· Scale factor for conversion. If **udf** or **AVS** is selected in the **file_type**, this parameter is not used.

- **IMPORT_LAMMPS...**

  Import the trajectory file (dump file) of "LAMMPS" output to **Strcuture** data. See the details of "LAMMPS" and its data format on the web page (http://lammps.sandia.gov/).

  This command is selectable at the root icon of tree structure.
  The arguments shown in Figure 7.10 are explained below.

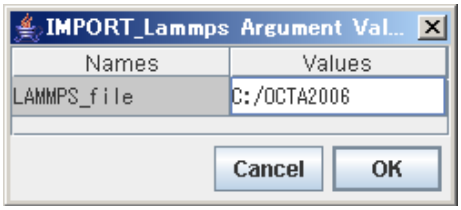

Figure 7.10: Arguments of **IMPORT_LAMMPS...**

  – **LAMMPS_file** ··· LAMMPS dump file name to import

- **INFO_Set_of_Molecules**

  Description : Output of the information of **Set_of_Molecules** such as the name of molecules, length of molecules.

  The informations are printed in a Python log window. This command is selectable at the root icon of tree structure.

- **EDIT_Make_Super_Cell...**

  Make a larger cell by multiplying unit cell. This command is selectable at the root icon of tree structure.

  The arguments shown in Figure 7.11 are explained below.

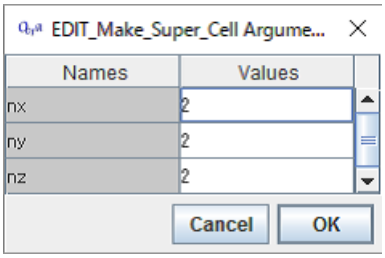

Figure 7.11: Arguments of **EDIT_Male_Super_Cell...**

  – **nx, (ny, nz)** ··· Expansion number of each cell length.

- **EDIT_Merge_Set_of_Molecules...**

  Description : Merger of **Set_of_Molecules** and **Structure** data of UDF file to current UDF data.

  This command is selectable at the root icon of tree structure.
  The arguments shown in Figure 7.12 are explained below.

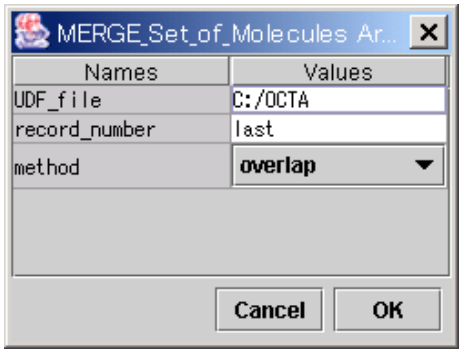

Figure 7.12: Arguments of **EDIT_Merge_Set_of_Molecules...**

- **UDF_file** ··· UDF file name to merge to current UDF.
  If you click input column with right button of mouse, file browser will appear.
- **record_number** ··· Record number of UDF file to merge.
  Record number(int) or keyword **last** (merger the last record) are accepted.
- **method** ··· Method of merger
  It is selected from **overlap/x_layer/y_layer/z_layer**. If **overlap** is selected, atoms of both UDF are packed in the unit cell of current UDF.
  If **x(,y,z)_layer** are selected, layered structure in specified direction are created. Unit cell length in layered direction will be extended to the sum of two unit cell.

This data processing affect only on the udf data in **GOURMET**, and the contents of the udf file is not changed untill the udf data on **GOURMET** is saved.

- **EDIT_Pack_Molecules...**

Description : Shift the coridnate of center of mass of each moleule in a unit cell. In the case that periodic bond is formed during polymerization for example, the cordinate of each atom is moved to prevent periodic bond.

This command is selectable at the root icon of tree structure.
The arguments shown in Figure 7.13 are explained below.

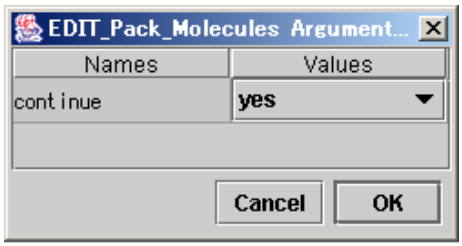

Figure 7.13: Arguments of **EDIT_Pack_Molecules...**

- **continue** ··· Confirmation of the execution of this action. The action is executed when **Yes** is selected.

This data processing affect only on the udf data in **GOURMET**, and the contents of the udf file is not changed untill the udf data on **GOURMET** is saved.

- **EDIT_Reduce_Molecules...**

  Description : Reduce the number of molecules (both **Set_of_Molecules** and **Structure**). This function can be used for zooming from the output of DPD to conventional bead-spring model for example.

  This command is selectable at the root icon of tree structure.
  The arguments shown in Figure 7.14 are explained below.

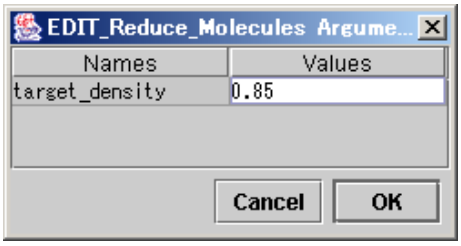

Figure 7.14: Arguments of **EDIT_Reduce_Molecules...**

  – **target_density** ··· Target density after reduced molecules. Currently the number density is used for scaling.

  This data processing affect only on the udf data in **GOURMET**, and the contents of the udf file is not changed untill the udf data on **GOURMET** is saved.

- **PLOT_data...**

  Description : Plotting of the data of specified UDF location in the sequence of records

  This command is selectable at the root icon of tree structure.
  The arguments shown in Figure 7.15 are explained below.

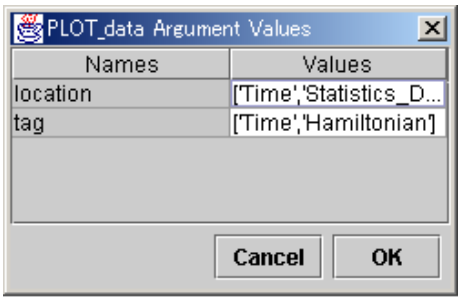

Figure 7.15: Arguments of **PLOT_data...**

  – **location** ··· A pair of UDF location, e.g.
  'Time','Statistics_Data.Energy.Batch_Average.Hamiltonian']
  The first location in the list will be x-axis, while the second one will be y-axis.

  – **tag** ··· List of tag for each location

- **PLOT_SS_curve...**

  Description : Plotting of Stress-strain (S-S) curve from the results of MD simulation with **Simple_Elongation**5.2.2.
  The arguments shown in Figure 7.16 are explained below.

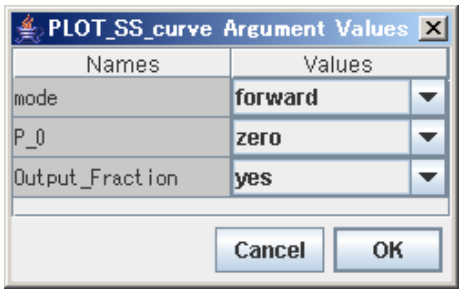

Figure 7.16: Arguments of **PLOT_SS_curve...**

  - **mode** ··· Selection of deformation mode at MD simulation. Elongation (**forward**) or compression (**reverse**). When **forward** is selected the initial record is referef for $strain_0$, while the final record is refered when (**reverse**) is selected.

  - **P_0** ··· The definition of hydrostatic pressure. When **zero** is selected, output stress value is the same as **stress_zz**. When **(xx+yy)/2** is selected, the stress output is defined as **Stress_zz - 1/2*(Stress_xx + Stress_yy)**.

  - **Ouput_Fraction** ··· The stress fractions from bonding and non bonding potential are output, when **yes** is selected.

- **VIEWER_show...**

  Description : Display of molecular structures in **Viewer** window

  This command is selectable at the root icon of tree structure.
  The arguments shown in Figure 7.17 are explained below.

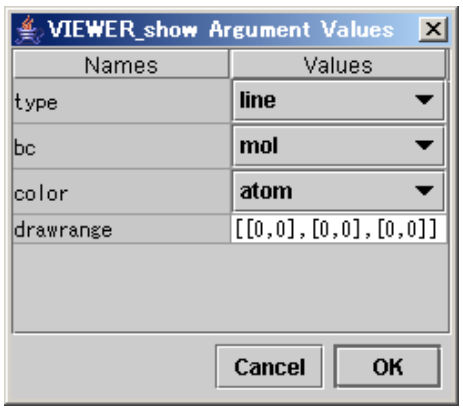

Figure 7.17: Arguments of **VIEWER_show...**

– **type** ··· Type of display
It is selected from **line/ball-stick/rod/volume**.

– **bc** ··· Boundary condition to display
It is selected from **mol/atom/off**.

– **color** ··· Type of color
It is selected from **atom/bond/mol/molname**.

– **drawrange** ··· Option of the range of display
The image molecules can be displayed in addition to those in basic unit cell. The range of the display is specified in the list as follows,
[[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
The number of cell to display in a,b and c axis is specified in *amin,amax,bmin...*, e.g. if the variable is set [[-1,1],[-1,1],[-1,1]], image molecules in the unit cells which are shifted by minus one and plus one for each axis is display in addition to the base (central) unit cell.

- **Viewer_show_field...**

Display of density distribution in **Viewer** window

This command is selectable at the root icon of tree structure.
The arguments shown in Figure 7.18 are explained below.

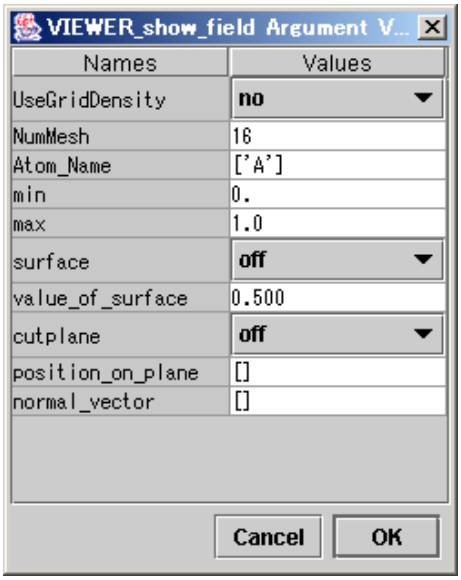

Figure 7.18: Arguments of **VIEWER_show_field...**

– **UseGridDensity** ··· Frag whether to use the grid density data in UDF.
It is selected from **yes/no**.
If **yes** is selected, grid density data is necessary in UDF. If **no** is selected, density distribution is calculated from the coordinate of atoms in Structure data.

– **NumMesh** ··· Number of grid point in one axis of unit cell.
If **UseGridDensity** is set to **yes**, the definition of mesh in Grid density data is used, and this value is ignored.

– **Atom_Name** ··· Atom type to display the density
List of Atom type is specified.

– **min** ··· Minimum value of color contour

– **max** ··· Maximum value of color contour

- **surface** ··· Frag whether to display color contour or iso density surface.
  It is selected from **yes/no**.
- **value_of_surface** ··· The value of iso density surface.
- **cutplane** ··· Frag whether to specify arbitrary plane for color contour.
  It is selected from **yes/no**. If **no** is specified, all plane of unit cell is displayed.
- **position_of_plane** ··· Position of plane for color contour.
  An arbitral position in a plane is specified as [**x,y,z**].
- **normal_vector** ··· Normal vector of the plane for color contour.
  It is specified as [**x,y,z**].

- **show...**

  Description : Display of objects in **Viewer** window

  This command is selectable at **Set_of_Molecules**, **molecule**, **atom** and **bond**. When specifying **molecule**, **atom** or **bond**, the index of each objects must be selected to launch the **Action** command. The arguments are almost the same as those of **VIEWER_show...**.

- **show_same_name...**

  Description : Display of molecules of the same name of molecules in **Viewer** window

  When an index of molecule is specified, all molecules, which have the same name of molecules, are displayed.
  The arguments are almost the same as that of **VIEWER_show...**.

- **show_same_type...**

  Description : Display of atoms of the same atom type in **Viewer** window

  When an index of atom is specified, all atoms, which is the same atom type, are displayed.
  The arguments are almost the same as that of **VIEWER_show...**.

- **plot**

  Description : Plotting of an object in **Statistics_Data** as a function of time

  This command is selectable at the objects in **Statistics_Data**. Simple objects such as **Temperature** are plotted by selecting the object. In the case of objects which have many items such as **Energy**, the plotting is possible at the objects, **Instantaneous**, **Batch_Average** and **Total_Average** separately.

- **{mol,atom,bond}_info**

  Description : Output of the information such as name and type of **molecule**, **atom** or **bond**

  The informations are printed in a Python log window by selecting an index of objects.

Next three **Action** commands are launched by picking multi atoms in **Viewer** window. Multi picking mode can be toggled on/off by selecting **Picking** menu on GOURMET **Viewer** window or typing **Ctrl+M**. See "GOURMET Operation manual" for the details.

Figure 7.19 shows an example of **Viewer** window at picking two atoms. A command **Distance...** is shown in the menu.

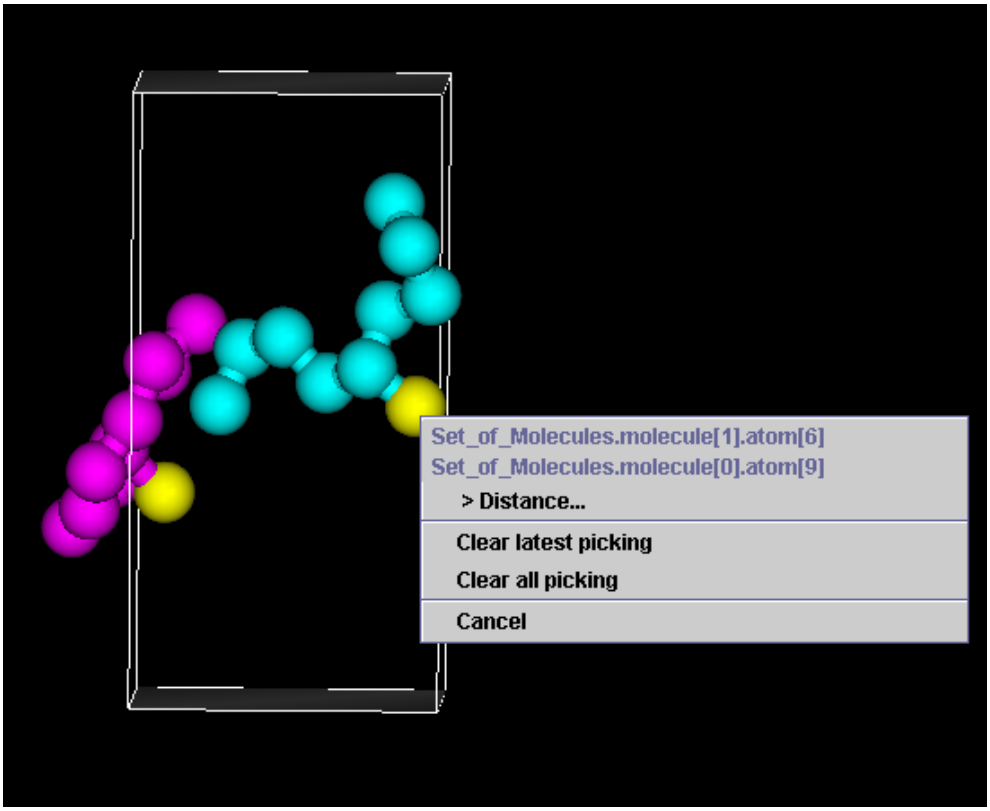

Figure 7.19: Multi picking

When three and four atoms are selected, **Angle** and **Torsion** are appeared in the menu respectively.

- **Distance**

  Description : Calculation of the distance between selected two atoms

  The arguments shown in Figure 7.20 are explained below.

  – **position** ··· Setting for boundary condition
  It is selected from **real/minimum**. When display is selected, the distance between the displayed atoms is calculated. If atoms are displayed with applying boundary conditions, the distance is calculated based on the image positions by boundary conditions. When minimum is selected, the distance of minimum image is calculated.

  – **draw** ··· Flag for displaying distance in **Viewer** window
  The distance is shown in the Viewer window as shown in Figure 7.21. The calculated distance is printed out in a Python log window in both case.

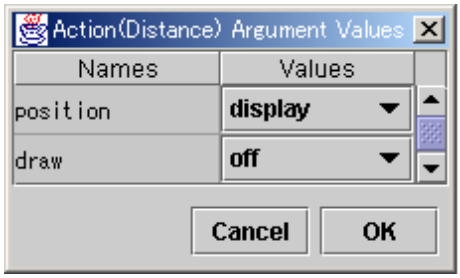

Figure 7.20: Arguments of **Distance...**

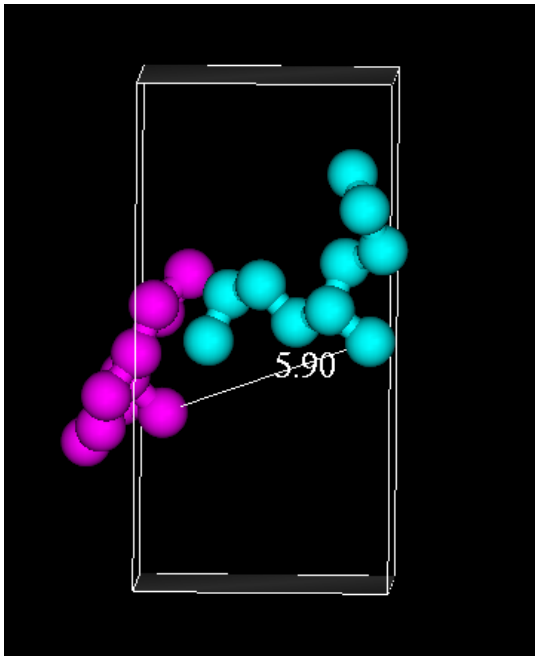

Figure 7.21: Display of distance

- **Angle**

  Description : Calculation of the angle of selected three atoms

  The angle value is printed out in a Python log window.

- **Torsion**

  Description : Calculation of the torsion angle of selected four atoms

  The torsion angle is printed out in a Python log window.

Appendix A

Compiling COGNAC

This appendix describes the procedure for compiling **COGNAC**. For example, the compilation is needed, when users add their own potential functions as it is shown in appendix B.

To make **COGNAC** object, GOURMET interface library "libplatform" is required. See the manual "Platform interface library 'libplatform' reference" for the detail of compiling environment.

1. MinGW/Cygwin/Linux/Unix environment

   "Makefile" in "src" directory can be used to compile **COGNAC**. Operating system and machine type are distinguished in "Makefile" based on an enviromental variable **$PF_ENGINEARCH**.
   Makefile will make OpenMP parallel version as a default. If you want to make non-parallel version, comment out the line 5, "OMP_FLAG =-fopenmp" in the Makefile.

   Procedure of compiling **COGNAC** using "Makefile"

   (a) Execution of **makedepend** (optional)

   If the **makedepend** command can be used in the environment to compile, run **makedepend** as follows.

   ```
   % make depend
   ```

   Notes:
   Although dependencies of ".cpp/.h" files are set in the "Makefile", it is recommended to perform **makedepend** in order to set all other dependencies. This is required only if source codes are modified, and it is unnecessary in the case of compiling existing files.

   (b) Compilation and linking

   The compilation and the linking of **COGNAC** are automatically performed using the default target of make.

   ```
   % make
   ```

   (c) Installation

   The executable file "cognac92{.exe}" generated by the compilation and linking is copied to a target directory "$(PF_ENGINE)/bin/$(PF_ENGINEARCH)" by

   ```
   % make install
   ```

$PF_ENGINE and **$PF_ENGINEARCH** are the environmental variables, which specify the directory where the simulation programs of OCTA are installed, and the environment (linux, cygwin etc.) where a simulation program is performed, respectively. They must be set up beforehand. Refer to the "OCTA installation guide" for details.

Notes:
"Makefile" can be used in some Unix enviroment. However, you need **gmake** to make **COGNAC** with included "Makefile". See "Makefile" for the detail of compile option.

2. Using Microsoft Visual C++ 2017

   The project file "COGNAC.vcxproj" for Microsoft Visual C++ 2017 is included in the directory "src/objects/win_vc2017". The compilation can be done using the project file. Visual C++ 2017 is supported in this release of COGNAC. Executable files are created in the "Release" or the "Debug" directory under "Win32" or "x64".

   Notes:
   To compile **COGNAC** with Microsoft Visual C++, the corresponding platform library must be build with Microsoft Visual C++ 2017. See the manual "Platform interface library 'libplatform' reference" for the detail.

Appendix B

Extension of potential functions

This appendix describes a procedure to implement user-defined potential functions and the usage of the functions.

B.1 User-definable potentials

In **COGNAC**, user-defined functions can be added to the following interaction potentials.

- Bond potential
- Angle potential
- Torsion potential
- Pair interaction
- External interaction

A procedure to implement new bond potential functions is explained, for example.

B.2 Needed files for user-defined potential

The following files are required in order to implement user-defined bond potential functions.

```
userbond[1-3].h
userbond[1-3].cpp
```

At the present, potential functions can be added simultaneously up to three, and the user-defined potential functions are prepared by the files "userbond{1-3}.h/.cpp". The template of the header files are prepared in the "include" directory. When creating other potential such as angle potential, the files which has a name "angle", "torsion", "pairinteraction", and "externalfield" should be prepared instead of "bond". See header files in the "include" directory for reference.

B.3 Implementation of new bond potential functions

B.3.1 Example of userbond1.h/.cpp

An example of user-defined potential "userbond1.h/.cpp" is shown. The files are contained in "include" and "src" directories. "userbond1" shown in the example defines bond potential of the same as the harmonic bond, $U_{bond} = 1/2k(r - r_0)^2$.

Example of userbond1.h

```
///// E-comment /////
// WG1
// file name "userbond1.h"
// version 1.0
// date 2000/11/22
// creator T.Aoyagi
// description
// These class (UserBond[1-3] are derived class from class Bond
//  for user defined bond potential.
//  Header files and a sample (userbond1.cpp) are included.
//
#ifndef _USERBOND1_H_
#define _USERBOND1_H_
#include "bond.h"
#include <map>
using namespace std;

class UserBond1 : public Bond {
public:
    UserBond1(map<string,string>& params,
        double r0=1.0, double rb=0.0, int i=0, string name="")
            : Bond(r0, rb, i, name) {
        kconst = atof(params["K"].c_str());
    }
    ~UserBond1() {}

    double calcForce(const Vector3d& dr, Vector3d& ftmp);
private:
    double kconst;
};

#endif //_USERBOND1_H_
```

Example of userbond1.cpp

```
///// E-comment /////
#include "userbond1.h"

double UserBond1::calcForce(const Vector3d& dr, Vector3d& ftmp)
{
    double r,delR,ene,tmp;

    r=dr.length();
    delR=r-r0;
    tmp=kconst*delR;
    ftmp=dr*(tmp/r);
    ene=0.5*tmp*delR;
    return(ene);
}
```

B.3.2 Member variables

The variables used in the potential function are defined as private members of “userbond1.h”. The example defines **double kconst**; which is a spring constant of the harmonic potential.

B.3.3 Description of Constructor

A constructor function must be defined and implemented in "userbond1.h" or "userbond1.cpp". In this constructor, all variables, which are required for the calculation of the potential function, are set from the arguments of constructor. The arguments of constructor are fixed and do not have to be changed. The variables set to **params** are actually passed to the private members of the class **UserBond1**. The object **params** is a **map** container of STL, and a data in the container is specified by **params**[**key**] like a usual array. The **param** used here takes string data as a key. If it is specified as **params**[**"K"**] as it is shown in the example, the data with key **"K"** is referred. And the variable data has also string type. Therefore, it is necessary to convert the type of the variable into proper type. For example, the variable **params**[**"K"**] must be converted to a double type variable **kconst** as follows.

```
kconst = atof(params["K"].c_str());
```

The method to specify the data with key by UDF files will be explained later. **COGNAC** takes an argument **params** which consists of combinations of key and data. And data is set to the private member in the constructor.

B.3.4 Description of calcForce

Since the calculation of potential is done by the method,

```
double calcForce(const Vector3d& dr, Vector3d& ftmp);
```

functional form of potentials should be defined here.

In the case of bond potential, **dr** and **ftmp** are taken as arguments, and the functions must be defined so that the calculated energy will be set as a return value. The bond vector $\mathbf{r}_1 - \mathbf{r}_0$ is passed to the argument **dr**. The **dr** is a object of class **Vector3d**, and has **double x, y, z** in members. The potential energy U and the force ($dU/d\mathbf{r}$) are calculated by using this **dr** and the private member, which is set by the constructor. The calculated U is returned as a double value. The force $dU/d\mathbf{r}$ can be passed to main program, by setting calculated value to **ftmp**. The **ftmp** is also a object of class **Vector3d**.
Since the arguments of **calcForce** of different potentials are different, the arguments of each potential are shown in the following. The return value is a potential energy in all cases.

```
double UserAngle[1-3]::calcForce(const Vector3d dr[],Vector3d ftmp[]);
```

- **dr**[]$\cdots$ Array of the bond vector which defines a bending angle.

  If a bending angle is defined as $\mathbf{r}_0 - \mathbf{r}_1 - \mathbf{r}_2$, **dr**[0] and **dr**[1] are set as $\mathbf{r}_1 - \mathbf{r}_0$ and $\mathbf{r}_2 - \mathbf{r}_1$ respectively.

- **ftmp**[]$\cdots$ Array of the force acting on the atoms, i.e.$dU/d\mathbf{r}_0$,$dU/d\mathbf{r}_1$ and $dU/d\mathbf{r}_2$

```
double UserTorsion[1-3]::calcForce(const Vector3d dr[],Vector3d ftmp[]);
```

- **dr**[]$\cdots$ Array of the bond vector which defines a torsion angle.

  If a torsion angle is defined as $\mathbf{r}_0 - \mathbf{r}_1 - \mathbf{r}_2 - \mathbf{r}_3$, **dr**[0],**dr**[1] and **dr**[2] are set as $\mathbf{r}_1 - \mathbf{r}_0$, $\mathbf{r}_2 - \mathbf{r}_1$ and $\mathbf{r}_3 - \mathbf{r}_2$ respectively.

- **ftmp**[]$\cdots$ Array of the force acting on the atoms $dU/d\mathbf{r}_0$,$dU/d\mathbf{r}_1$,$dU/d\mathbf{r}_2$ and $dU/d\mathbf{r}_3$

```
double UserPairInteraction[1-3]::calcForce(const Vector3d r[], Vector3d ftmp[]);
```

- **r**[]$\cdots$ Array of the position of atoms which defines an interaction site
  For example, if a pair interaction is applied between interaction sites defined by one atom, the positions of two atoms are set in **r**[**0**] and **r**[**1**], and if a pair interaction is applied between interaction sites defined by two atoms, the positions of two atoms which define first site are set in **r**[**0**] and **r**[**1**], and the other two atoms which define second site are set in **r**[**2**] and **r**[**3**].
- **ftmp**[]$\cdots$ Array of the force acting on the atoms, $dU/d\mathbf{r}_i$
  The indexes of the array **ftmp**[] corresponds to those of **r**[].

```
double UserExternalField[1-3]::calcForce(
        const Vector3d r[], Vector3d ftmp[], SymMat3x3 vtmp[],Vector3d wtmp[]);
```

- **r**[]$\cdots$ Array of the position of atoms which defines an interaction site
- **ftmp**[]$\cdots$ Array of the force acting on the atoms, $dU/d\mathbf{r}_i$
  The indexes of the array **ftmp**[] corresponds to those of **r**[].
- **vtmp**[]$\cdots$ Array of the virial term $-\mathbf{r}_i\mathbf{f}_i$
  This array is used to calculate pressure, and the indexes of the array **vtmp**[] corresponds to those of **r**[]. Each element of **vtmp**[] is a object of class **SymMat3x3**, and has **double xx, yy, zz, yz, zx, xy** in members.
- **wtmp**[]$\cdots$ Array of the force acting on the wall
  When a solid wall is set as an external field, the array **wtmp**[] is set to calculate the pressure acting on the wall. The indexes of the array **wtmp**[] corresponds to those of **r**[].
  For example, if the walls are placed in the position of $z = 0$ and $Zmax$ on xy plane, **wtmp**[**i**] corresponding to **r**[**i**] is given by $1/2\,\{dU_{wall\,at\,z=0}(\mathbf{r})/d\mathbf{r} - dU_{wall\,at\,z=Zmax}(\mathbf{r})/d\mathbf{r}\}$.
  You may ignore this argument, when the external potential is not the potential of wall, or when the pressure concerning walls does not need to be calculated.

B.3.5 Description of UserPairInteraction[1-3]::calcDragForce

A method **calcDragForce** in the class **UserPairInteraction**[**1-3**] will calculate pair potential from the position and velocity of atoms, e.g. fluctuation and dissipation force in DPD. The arguments and return value are following.

```
double UserPairInteraction[1-3]::calcForce(const Vector3d r[], const Vector3d v[],
Vector3d fdissipative[], Vector3d frandom[])
```

- **r**[] $\cdots$ Array of the position of atoms which defines an interaction site For example, if a pair interaction is applied between interaction sites defined by one atom, the positions of two atoms are set in **r**[**0**] and **r**[**1**], and if a pair interaction is applied between interaction sites defined by two atoms, the positions of two atoms which define first site are set in **r**[**0**] and **r**[**1**], and the other two atoms which define second site are set in **r**[**2**] and **r**[**3**].
- **v**[] $\cdots$ Array of the velocity of atoms which defines an interaction site. The indexes of the array **v**[] corresponds to those of **r**[].
- **fdissipative**[] $\cdots$ Array of the dissipative force acting on the atoms. The indexes of the array **fdissipative**[] corresponds to those of **r**[].
- **frandom**[] $\cdots$ Array of the random force acting on the atoms. The indexes of the array frandom[] corresponds to those of **r**[]. Note that **fdissiaptive**[] is include in the Virial term for calculating pressure tensor, while **frandom**[] is not.
- **return value** $\cdots$ Potential energy (This value won ' t be used in usual case.)

B.3.6 Description of UserPairInteraction[1-3]::calcTailCorrection

A method **calcTailCorrection** can be used for performing tail correction of the potential energy and the virial term in the class **UserPairInteraction[1-3]**. The contents of **calcTailCorrection** in "userpairinteraction1.cpp" in src" directory is shown, for example.
In this example, the tail correction of the Lennard-Jones potential is performed as shown in the reference [**?**].

```
SymMat3x3 UserPairInteraction1::calcTailCorrection(double& ene, int n1,
        int n2, const double volume)
{
    SymMat3x3 tailVirial(1.0,1.0,1.0);
    double tailVir;
    double temp1,temp2,temp3;
    double sigsix,sigtwelve,rcnine,rcthree;

    sigsix = sigmaSq*sigmaSq*sigmaSq;
    sigtwelve = sigsix*sigsix;
    rcthree = cutOff*cutOff*cutOff;
    rcnine = rcthree*rcthree*rcthree;

    temp1 = 8.0*PI*epsilon*(double)n1*(double)n2/(3.0*volume);
    temp2 = sigtwelve/(rcnine*3.0);
    temp3 = sigsix/rcthree;

    tailVir = -2.0*temp1*(2.0*temp2 - temp3);
    tailVirial *= tailVir;

    ene = temp1*(temp2 - temp3);

    return tailVirial;
}
```

Arguments and return value are following.

- **ene** $\cdots$ Corrected potential energy
- **n1, n2** $\cdots$ Number of interaction sites in a system
  If the pair interaction is calculated between the same sites, set **n1=n2**.
- **volume** $\cdots$ Volume of a unit cell
- **return value** $\cdots$ Corrected virial term
  The return value is an object of class **SymMat3x3** (**tailVirial** in the example), which has **double xx, yy, zz, yz, zx, xy** in members.

And the following member variables are currently used in the example.

- **cutOff** $\cdots$ Cutoff distance
  This is a member of the base class **PairInteraction**
- **PI** $\cdots$ Value of PI $\pi = 3.1415....$
  This is defined as a global parameter of **COGNAC**.
- **sigmaSq, epsilon** $\cdots$ Private members of the class **UserPairInteraction1**

B.3.7 Description of UserExternalField[1-3]::setCell

In **UserExternalField[1-3]**, users can implement the potential functions, which uses the unit cell size to calculate the functions: e.g. the potential functions may use the volume or the surface area of unit cell. The method **setCell** will update the information of the unit cell in the class **UserExternalField[1-3]** every time step.
The example shown in the following is a part of the Lennard-Jones type flat wall implemented in **COGNAC**, and in the case of non-rectangular cell, the process will terminate with showing error message.

```
void LJWall::setCell(const SymMat3x3& cell){
    if(cell.yz != 90.0 || cell.zx != 90.0 || cell.xy != 90.0 ){
        throw NotSupport("LJ_Wall potential to non-rectangular cell");
    }
    c_shape = cell;
}
```

The argument **cell** is an object of class **SymMat3x3** which has **double xx, yy, zz, yz, zx, xy** in members, and the unit cell size $a, b, c, alpha, beta, gamma$ are set to the argument (the unit of angles is degree). The **c_shape** used in the example is a private member of a class **ExternalField**, and it is also an object of class **SymMat3x3**.

B.4 Recompilation of COGNAC

When users add their own potential functions, it is necessary to recompile **COGNAC**. Refer to appendix A for the method of general compilation.
Notice that it is necessary to add the files to the workspace in Visual studio or "Makefile", when the files are newly created.

B.5 Input of parameters in UDF

The method to use the user-defined potential function in input UDF is explained. The UDF path for user-defined potentials has parameters **Index**, **Param[].Name** and **Param[].Value**. The description of the parameters are as follows.

- **Index** $\cdots$ Index of the class name of user-defined potential.

  For example, if it is specified as **Index=1** in **User_Bond**, the class **UserBond1** is used as the potential. Therefore, this index takes an integer of 1-3.

- **Parameter[].Name** $\cdots$ Key of map **params** which is a constructor's argument.

  Arbitrary string data which specifies the data of **params** is given as key.

- **Parameter[].Value** $\cdots$ Data of map **params** which is a constructor's argument.
  Since all variables are given as string data in input, users have to carry out type conversion in the constructor mentioned above, if it is needed.

The array **Parameter[]** have elements of private variables used by the user-defined potential function.

Appendix C

Import of molecular data from other format

In addition to build molecular architecture by **SILK**, **COGNAC** can import molecular data, i.e. molecular architecture and coordinate, from other format data using **MoleculeBuilder**, which is a function of **GOURMET**. This Appendix explains the example how to import mol formatted file, which is one of the common file format to describe molecular data. The sample molecule is one molecule of tetradecane, ($C_{14}H_{30}$), united atom model.

- This example will explain only how to import the topological data and coordinates.
- If you are working with Linux environment, you should convert the name of directory to appropriately.

1. Preparing the input UDF of COGNAC

   First, make a working directory. (here, "C:\work_octa" is created.)
   Then, copy "c14.mol" in the directory "sample\molfile_sample"
   to the working directory. Next, create the file "test.udf" by text editor.
   Write the following line in the first line of the file,

   ```
   \include {"cognac92.udf"}
   ```

   then, save as "c:\work_octa\test.udf".

2. Open "test.udf" in **GOURMET**

   Start **GOURMET**, and open "test.udf" (**File**→ **Open** from the menu of **GOURMET**)

3. Import the file

   Select **Tool** → **MoleculeBuilder** from the menu of **GOURMET**, then input the following information in the appeared pop-up menu.

   ```
   Select File Filter Type: molfile format Filter
   Data File: c:\work_octa\c14.mol
   Input UDF: c:\work_octa\test.udf
   Output UDF: c:\work_octa\test.udf
   ```

   - **Data File** is an original file name to be imported.
   - Set the same name "test.udf" to both **Input UDF** and **Output UDF** in this example.
   - Keep the check box of **GenerateElectrostatic Site OFF**.

Next, select **OK**. Also, select **Yes** to the message **Disregard Current Changing?**.

Then, the molecular data is imported to the **Set_of_Molecules** and **Structure** of "test.udf".

4. Display the imported data

   Display the molecular structure by using **VIEWER_show** in **GOURMET**. (See the section "Display of molecules using **Action**" in the chapter 3.)

   - Set the parameter **bc** to **off** in the appearing window. Otherwise the molecular structure will not be displayed properly, since there is no boundary conditions in the imported data.

   One molecule of tetradecane united atom model (all trans conformation) will be displayed, if the import is done.