OCTA

ソフトマテリアルのための統合化シミュレータ

汎用粗視化分子動力学シミュレータ

COGNAC

ユーザーズマニュアル

Version 9.2

OCTA ユーザーズグループ

Dec 25 2017

執筆者

第 1 章　青柳岳司
第 2 章　青柳岳司
第 3 章　青柳岳司、澤史雄
第 4 章　青柳岳司、澤史雄
第 5 章　青柳岳司
第 6 章　澤史雄
第 7 章　青柳岳司
付録 A　青柳岳司
付録 B　青柳岳司
付録 C　青柳岳司、澤史雄

プログラム開発者

青柳岳司
澤史雄
庄司達也
福永宏雄

謝辞

本プログラムは、経済産業省の出資 · 補助を受け、新エネルギー · 産業技術総合開発機構 (NEDO) が (財) 化学技術戦略推進機構に委託した、大学連携型産業科学技術研究開発プロジェクト「高機能材料設計プラットフォーム」通称「土井プロジェクト」の下で開発され、プロジェクト終了後 2002 年 4 月より、OCTA ユーザーズグループにより開発 · メンテナンスが継続されている。

目 次

図 目 次

表 目 次

第1章　COGNACとは

COGNAC (COarse Grained molecular dynamics program by NAgoya Cooperation) とは、高機能材料設計プラットフォームの研究開発の一環として開発された、汎用粗視化分子動力学プログラムである。[1]

粗視化分子動力学とは、原子1個を一つの質点として運動方程式を解いていく、通常のアトミスティックな古典分子動力学と異なり、いくつかの原子の集合体を一つの単位として、その構成単位のダイナミクスをシミュレーションする。それにより通常のアトミスティックな分子動力学に比較して、大きなサイズ（原子数、分子数、重合度）のシステムの長時間のダイナミクスをシミュレーションすることが可能になる。ただし、シミュレーションモデルの自由度が大きくなる分、粗視化の単位、粗視化単位のポテンシャル関数、およびポテンシャルパラメータなどの選択の自由度も増大し、ユーザーにゆだねられる部分が大きい。

COGNAC においては、分子動力学として一般的に用いられる各種アンサンブル、およびポテンシャル関数に加えて、ビーズ－スプリングモデル、Dissipative particle dynamics(DPD) 等の粗視化シミュレーションを行う際に利用される、ポテシンャル関数、運動方程式が組み込まれている。また材料物性予測のために有用な、流動、伸張変形、あるいは固体壁などの外場、架橋反応等の化学反応などの機能を有する。さらにユーザーによるモデル、ポテシンャル関数等の拡張が容易に行えるプログラム構成を持つ、汎用粗視化分子動力学プログラムとして、設計、開発が行われた。

これら多様な機能に加えて、**COGNAC** は相分離構造を取り扱うために、新たに開発されたアルゴリズムを持ち、平均場シミュレータ **SUSHI** により求められた相分離構造に基づいて、粗視化分子動力学のための分子鎖構造を生成し、シミュレーションを行うことが出来る。この機能により、**SUSHI**、**Muffin** などの他のメソスケールシミュレータとの連携が可能になり、メソ領域シミュレーションによる高機能材料設計に対して、有用なツールとなる。

2014 年 3 月、**OCTA** および **COGNAC** はじめ **OCTA** に含まれる他のエンジンの概要、適用事例を紹介する書籍が発行され [2]、2017 年 7 月には増補版が発行された [3]。OCTA を活用するための有益な内容が含まれているので、マニュアルとあわせてそちらの書籍も参照されたい。

第2章　理論背景

本章では、**COGNAC** の主な機能、および **COGNAC** で用いている（粗視化）分子動力学シミュレーションのためのアルゴリズムの理論的背景を解説する。機能の詳細に関しては 5 章を参照のこと。

なお、分子動力学のアルゴリズム全般に関しては参考文献 [4, 5, 6, 7, 8, 9, 10, 11] 等も参考にされたい。

2.1　機能概要

以下に、**COGNAC** の主な機能をあげる

- 計算可能な分子構造：任意のアーキテクチャー（直鎖および分岐高分子鎖、低分子、単原子分子等) の分子構造の集合
- 初期構造作成：非晶、結晶、半結晶ラメラ、**SUSHI** によって得られた相分離構造などの初期座標を生成
- 分子動力学アンサンブル：NVE、NVT、NPH、NPT、および Lees-Edwards 境界条件によるせん断流動、あるいは Unit Cell の変形による非平衡分子動力学
- 分子力学：最急降下法、共役勾配法等によるエネルギー極小化
- DPD：Dissipative particel dynamics [12, 13]
- ポテンシャル：結合ポテンシャル (Bond、Angle、Torsion)、非結合ポテンシャル、静電力、外場（固体反射壁、均一電場、濃度場ポテンシャル等）
- 境界条件：周期境界条件。Lees-Edwards 境界条件。スタガード反射境界条件
- 化学反応：化学結合の生成、切断、重合反応、原子種の置換

2.2　注意事項

ここでは **COGNAC** の機能説明上の注意、用語の定義などを数点あげる

2.2.1　単位系

COGNAC の入出力は無次元単位系を用いている。実単位とのコンバートを行う際には、長さ、質量、時間等異なる次元を持つ 3 種類のパラメータの reduced unit と real unit の対応関係を与えて換算する。**COGNAC** UDF データにスケーリングパラメータをセットすることにより、GOURMET 上で単位換算を用意に行うことが出来る。詳細は §5.2.9 および「GOURMET 操作マニュアル」参照。

無次元単位系の換算例（United atom の例）

Reduced Length $1L = 0.38\text{nm}$
Reduced Energy (Energy density) $1\epsilon = 0.5\text{kJ/mol}$
Reduced Mass $1M = 23.3 \times 10^{-27}\text{kg}$
とおいた場合（これらの値は United atom model における CH_2 unit の質量と、Lennard-Jones ポテンシャルより決めている [14]）

Reduced Temperature $1T = \epsilon/\text{R} = 60.1\text{K}$
Reduced Time $1t = \sqrt{\text{Av}ML^2/\epsilon} = 2.01\text{ps}$
Reduced Density $1\rho = M/L^3 = 0.4246\text{g/cm}^3$
Reduced Pressure $1P = \epsilon/(\text{Av}L^3) = 15.13\text{MPa}$

R:気体定数, Av:アボガドロ数

2.2.2 COGNAC における圧力とストレス

COGNAC においては圧力（Pressure, P）とストレステンソル（Stress, σ）は異符号と定義している。すなわち系の圧力 P と外圧 P_0 の関係が $P - P_0 > 0$ の場合は、系は膨張する方向を示すが、系のストレス σ と外部より与えるストレス σ_0 の関係が $\sigma - \sigma_0 > 0$ の場合は系は収縮する方向を示す。よって、計算結果として出力される値は $P = -\mathbf{Tr}\sigma/3$ となる。

また、実際に NPT アンサンブル等で圧力、ストレステンソルを指定する場合（§5.2.2 参照）、$P_0\mathbf{I} - \sigma_\mathbf{0}$ の圧力テンソルが系に外圧として与えられる。

2.2.3 COGNAC における結合角と二面角の定義

COGNAC においては、高分子における結合角と二面角の定義でよく用いられる定義に基づき、**結合角においては外角、すなわち直線分子の結合角を 0°**、また**二面角においてはトランスを 0°** としている。また入力 UDF 等で指定する角度は degree を用いる。

2.2.4 用語の定義

原子、atom

粗視化分子動力学の場合、1 つの質点が 1 原子を表すことには必ずしもならないが、本マニュアルでは、便宜上すべての質点を表す用語として”原子”、あるいは”atom”を用いる。

Interaction Site

COGNAC の特徴として Pair interaction あるいは External interaction を作用させるサイトを原子とは別に定義する。そのサイトを”Interaction Site”と呼び、後述する方法により図 2.1 に示すように 1 つあるいは複数の atom により Interaction Site を定義することができ、定義に用いた atom すべての座標を用いて Pair interaction を定義することができる。

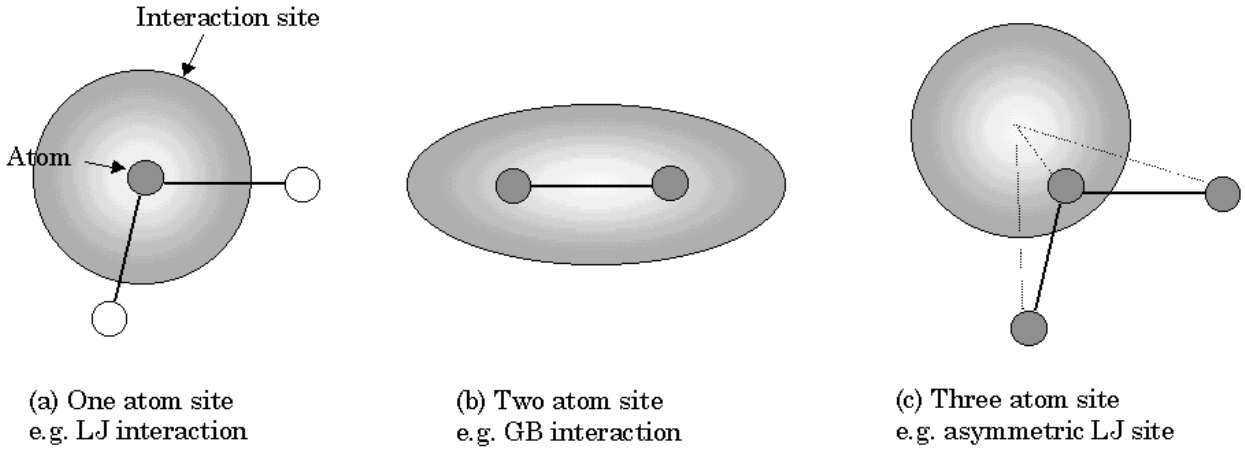


図 2.1: Interaction Site の定義例

Electrostatic Site

静電相互作用を計算するサイトとして、Interaction Site と同様に原子とは別に定義する。そのサイトを ”Electrostatic Site”と呼び、点電荷の場合は 1 つの atom、双極子の場合は 2 つの atom により定義される。

2.3 分子動力学

実際にシミュレーションを行う分子動力学部分に用いられている、アルゴリズムの解説を行う。

2.3.1 温度と圧力の計算

分子動力学シミュレーションにおいて、システムの温度および圧力テンソルは以下のように計算される。

- 温度 T

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

ここで N は原子数、k_B はボルツマン定数、$\mathbf{p}_i$、m_i は各々、原子 i の運動量と質量である。N_c は拘束される自由度の数で、たとえば 3 次元周期境界条件の場合は $N_c = 3$ となる。同時に §2.3.4 で述べる拘束条件を課した場合は、その分 N_c としてカウントされる。

- 圧力テンソル $\mathbf{P}$ 周期境界条件あるいは、§2.8 にあるスタガード反射境界条件を用いる場合、圧力テンソル $\mathbf{P}$ は下式で計算される。

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

ここで、V はシステムの体積、N は原子数、$\mathbf{p}_i$、m_i は原子 i の各々の運動量と質量、$\mathbf{r}_{ij}$ は原子 i, j 間ベクトル $\mathbf{r}_i - \mathbf{r}_j$、$\mathbf{f}_{ij}$ は原子 j が原子 i に及ぼす力である。

2.3.2 アンサンブル

- マイクロカノニカルアンサンブル (NVE ensemble)

  ポテンシャルエネルギー $P.E$ と運動エネルギー $K.E$ の和で定義されるハミルトニアン $H = P.E + K.E.$ が一定に保たれるようなアンサンブル。**COGNAC** においては NVE の指定でも 2.3 式による温度スケールにより温度を制御することは可能

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

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

- 温度制御法 (NVT ensemble)

  COGNAC は前述の温度スケール以外に温度を制御する方法として以下のアルゴリズムを持つ

  - Nose-Hoover [15]

    2.4 − 2.6 式に従い、仮想質量 Q を与えて温度制御を行う。

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

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

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

    $\mathbf{q}$:real coordinate of atom, $\mathbf{p}$:moments, U:potential, g:degree of freedom
    T:target temperature, Q:fictitious mass

  - Loose-Coupling [16]

    2.7 式で得られる、λ により速度をスケールする。

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

    Δt:time step, τ:characteristic relaxation time, T_0:target temperature, T:current temperature

  - Kremer-Grest (Langevin-Dynamics) [17]

    2.8 式の運動方程式を用いた Friction constant + Thermal noise による温度制御。Langevin-Dynamics と等価

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

    $\boldsymbol{r}_n$:原子 n の座標, m_n:質量, $\boldsymbol{F}_n$:力, Γ:friction constant, $\boldsymbol{W}_n(t)$:2.9 式で定義される Gaussian white noise

$$\langle \boldsymbol{W}_n(t) \boldsymbol{W}_m(t') \rangle = 2 k_B T m_n \Gamma \Delta_{nm} \boldsymbol{I} \Delta(t - t') \tag{2.9}$$

- 圧力制御法 (NPH ensemble)

  COGNAC は圧力を制御する方法として以下のアルゴリズムを持つ

  - Andersen [18]

    拡張ハミルトニアン法による圧力制御。Lagrangian L は 2.10 式で表され、運動方程式は 2.13、2.14 式より計算される。

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

    $\mathcal{K}$: 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.
    $\mathcal{K}_V$ および $\mathcal{V}_V$ は以下で定義される。

    $$\mathcal{K}_V = \frac{1}{2}Q\dot{V}^2 \tag{2.11}$$
    $$\mathcal{V}_V = P_0 V \tag{2.12}$$

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

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

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

  - Parrinello-Rahman [19]

    Andersen アルゴリズムを拡張し、圧力テンソルの個々の成分を制御する。よってユニットセルの異方的な変形が可能。Lagrangian L は Andersen のアルゴリズムと同様に 2.15 式で表され (external stress の無い場合)、運動方程式は 2.18、2.19 式より計算される。

    $$\mathcal{L} = \mathcal{K} + \mathcal{K}_V - \mathcal{V} - \mathcal{V}_V \tag{2.15}$$

    $\mathcal{K}$: 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.

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

    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.

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

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

    ユニットセル角度を固定する場合は、2.18 式の $\ddot{\mathbf{s}}$ の非対角成分を強制的にゼロにして計算している。この場合ハミルトニアンは保存されない可能性がある。

- Loose-Coupling [20]

  Loose-Coupling 法により圧力テンソルの個々の成分を制御する。すなわち Parrinello-Rahman と同様ユニットセルの異方的変形が可能。

- 温度－圧力制御法 (NPT ensemble)

  COGNAC では基本的に前述の NPT、NPH のいくつかの組み合わせのアルゴリズムが利用できる。

  - Andersen + Nose-Hoover

    Andersen 拡張ハミルトニアン法による圧力制御　+ Nose-Hoover 法による温度制御。

  - Andersen + Kremer-Grest

    Andersen 拡張ハミルトニアン法による圧力制御　+ Kremer-Grest 法による温度制御。

  - Parrinello-Rahman + Nose-Hoover

    Parrinello-Rahman 拡張ハミルトニアン法による圧力制御　+ Nose-Hoover 法による温度制御

  - Parrinello-Rahman + Langevin dynamics

    Parrinello-Rahman 拡張ハミルトニアン法による圧力制御　+ Langevin dynamic による温度制御。

  - Loose-Coupling [16]

    Loose-Coupling 法による圧力、温度制御。ユニットセル等方／非等方変形が可能圧力制御は等方的制御の場合、2.20 式で得られる μ により、原子の座標とユニットセルサイズをスケールする。

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

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

    非等方的制御の場合は、2.21 式に従い transformation matrix$\mathbf{H}$ の時間発展を求め、原子の座標をスケールする。

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

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

    ユニットセル角度を固定する場合は、2.21 式の $\dot{\mathbf{H}}$ の非対角成分を強制的にゼロにして計算している。

- 非平衡ダイナミクス

  COGNAC においては流動、変形などのシミュレーションを行う方法として次の 2 通りの方法をもつ。

Lees-Edwards 境界条件によるせん断流動

NVE あるいは Langevin dynamics において、Lees-Edwards 境界条件 [21] を用いてせん断流動を与えることができる。
NVE+温度スケールあるいは Langevin dynamics により流動下において温度制御を行うことも可能であるが、SLLOD+Nose-Hoover 法による温度制御 [22] も可能である。

一定速度のせん断流動を与える場合の SLLOD の運動方程式は 2.22-2.27 式のようになる [4]

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

$r_{i\alpha}$:coordinate of atom i, $p_{i\alpha}$:moment of atom i, $f_{i\alpha}$:force on atom i

また、COGNAC では温度制御に加えて応力の制御を行うことも可能である。方法としては Parrinello-Rahman 法により法線方向の応力を制御する。すなわち $\dot{\gamma}_{yx}$ の場合、z 方向の応力を制御するためユニットセル c 軸のみ変化する。

（注） Version 8.0 より SLLOD の際の温度制御は拘束法より Nose-Hoover 法に、圧力制御は Loose-Coupling 法より Parrinello-Rahman 法に変更になった。

外圧あるいはセルの動的変形

通常のアンサンブルによるシミュレーションの過程で、外圧あるいはユニットセルを動的に変形させることにより、非平衡状態のダイナミクスをシミュレートすることができる。

- 外圧を与えたシミュレーション

  Parrinello-Rahman あるいは Loose-Coupling による圧力制御を行うアンサンブルを選択し、等方的な圧力に加えて、外圧に対応する応力テンソルを与えると外圧に応じてユニットセルが変形してしき非平衡状態のシミュレーションを行うことができる。

- ユニットセルの変形を伴ったシミュレーション

  通常の MD シミュレーションの過程で、ユニットセルを一定のタイムステップ毎に変形させる。このユニットセルの変形の際同時に全原子の座標をユニットセルの変形に対応させてアフィン変形をおこなう。この変形間隔はあまり長く取りすぎると一時の変形量が大きくなりすぎシミュレーションが不安定になる。逆に短いとユニットセルサイズのリセットなどのための計算時間が多くかかるので、経験的に最適間隔を決めてやる必要がある。
  また、変形の方法としては、変位テンソルを与える方法、あるいは単純な伸張変形を与える方法などがある。

2.3.3 Integration algorithm

COGNAC においては MD の時間発展に velocity Verlet algorithm[23] を使用している。一般的な MD の時間発展としては [23] 等に解説されているように、この velocity Verlet 以外にも、いくつか方法はあるが、粗視化分子動力学に要求されるハミルトニアン保存の精度、タイムステップの変化による安定性などを考慮して **COGNAC** ではこのアルゴリズムを採用している。

velocity Verlet において、座標 $\mathbf{r}_i$ および速度 $\mathbf{v}_i$ の時間発展は 2.28-2.30 式で記述される。

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

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

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

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

2.3.4 Constraint

- RATTLE

  RATTLE アルゴリズム [24] により結合長、および結合角を拘束する機能をサポートしている。

- Constraint Atom

  任意の原子の運動を拘束する機能をサポート。
  実際のシミュレーションにおいては、指定した原子の力を強制的にゼロにし、同時に一定の速度を与えて時間発展を計算する。この場合、ポテンシャルエネルギー、およびペアポテンシャルの場合、ペアの原子にかかる力はゼロにしない。
  この場合、ハミルトニアンは保存されない可能性があるので注意。

2.3.5 Energy Flow

COGNAC は MD シミュレーションの過程におけるエネルギーの出入りを計算する機能を持つ。現バージョンでは、計算が可能なのは NVE+温度スケールの場合のみであるが、非平衡 MD を行った場合の吸発熱量の見積もりを行うことが可能である。

計算の方法としては単純で、2.3 式にしたがって温度スケールを行った際、スケール前後の温度 T_{system}、T_{target} から、2.31 式で求められるエネルギーの出入りをスケールした回数積算して、区間積算量と全積算量を求める。

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

N: degree of freedom

この定義より、発熱が正の値として出力される。

2.4 DPD

2.4.1 運動方程式

DPD においても、Atom i の運動は Newton の運動方程式により記述される。

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

Atom に作用する力 $\mathbf{f}_i$ は以下のように 3 項よりなる。

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

ここで、$\mathbf{F}_{ij}^C$ は通常の粒子間相互作用、$\mathbf{F}_{ij}^D, \mathbf{F}_{ij}^R$ は 2.34 式で表される散逸項とランダム力である。

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

$\theta_{ij}(t)$ は、$\langle \theta_{ij}(t) \rangle = 0$ および $\langle \theta_{ij}(t)\theta_{kl}(t') \rangle = (\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})\delta(t - t')$ を満たす、ランダムノイズである。

その他の変数の関係は Groot らの論文 [12] に基づき、以下のように決められる。

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

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

DPD における時間積分は、2.37-2.40 式に示すように velocity Verlet を修正した形式をとる。[12]

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

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

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

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

λ : variable factor

2.5 分子力学

COGNAC で用いられている分子力学における最適化のアルゴリズムをあげる。最適化アルゴリズムの詳細は諸々の成書を参考にされたい。

2.5.1 最適化アルゴリズム

- 最急降下法
- 共役勾配法
- カスケード
  最急降下法と共役勾配法の組み合わせ。最適化初期において最急降下法を用い、ポテンシャル勾配が低下した時点で自動的に共役勾配法に切り替える。

2.6 ポテンシャル

COGNAC は以下のような相互作用ポテンシャルを計算することにより、系のポテンシャルエネルギーおよび個々の原子に作用する力を計算する。

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

U_{bond}：結合伸縮（2 体間）ポテンシャル
U_{angle}：結合変角（3 体間）ポテンシャル
$U_{torsion}$：結合 2 面角（4 体間）ポテンシャル
$U_{non-bonding}$：非結合（原則 2 体間）相互作用
$U_{coulomb}$：静電相互作用（原則 2 体間）
$U_{external}$：　外場によるポテンシャル

以下に、各項において利用できる関数形等を解説する。

2.6.1 結合伸縮ポテンシャル

結合を定義した原子間の距離に応じて変化するポテンシャル。**COGNAC** では以下の種類が利用できる。

- 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 [25]

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

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

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

（注意）Version7 以下では FENE+LJ のエネルギーの出力値を表示が見やすいように任意の値でシフトしていたが、Version8 以降は表式どおりの値を出力するように変更した

- 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
  - エネルギーの最小値を 0 にする場合

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

– 無限遠のエネルギーを 0 にする場合

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

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: constant

- Table

結合長 r あるいは r と温度 T の組み合わせを変量とするエネルギー値 U_{bond} のテーブルデータにより、任意の結合長と NVT アンサンブルなどにおいて設定した温度におけるエネルギーと力を内挿して計算する。ただし Fast_Table を用いた場合、距離のみの関数となる。

- DPD

Groot らの [12, 13] 報告する DPD において、用いられている Bond potential。基本的には Harmonic potential と同じ

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

C: constant

- User Bond

ユーザーがソースコードレベルで定義したポテンシャル関数を用いることができる。

2.6.2 結合変角ポテンシャル

定義した 3 つの原子のなす角度に応じて変化するポテンシャル。**COGNAC** では以下の種類が利用できる。

- Theta harmonic

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

k: spring constant, θ_0: equilibrium angle

- Theta harmonic 2

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

k: spring constant, θ_0: equilibrium angle
Theta harmonic において $\theta = 0 \quad or \quad 180 deg.$ になった場合、力の計算が発散するので、その場合はこの表式を用いるとよい

- 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: constant

- Table

結合角 θ の場合 $cos\theta$、あるいは $cos\theta$ と温度 T の組み合わせを変量とするエネルギー値 U_{angle} のテーブルデータにより、任意の結合角と NVT アンサンブルなどにおいて設定した温度におけるエネルギーと力を内挿して計算する。ただし Fast_Table を用いた場合、結合角のみの関数となる。

- User Angle

ユーザーがソースコードレベルで定義したポテンシャル関数を用いることができる。

2.6.3 結合二面角ポテンシャル

定義した 4 つの原子のなす 2 面角に応じて変化するポテンシャル。**COGNAC** では以下の種類が利用できる。

- 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: constant

- Amber [26]

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

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

- Dreiding [27]

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

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

- Table

  結合二面角 ϕ の場合 $cos\phi$ あるいは $cos\phi$ と温度 T の組み合わせを変量とするエネルギー値 $U_{torsion}$ のテーブルデータにより、任意の結合二面角と NVT アンサンブルなどにおいて設定した温度におけるエネルギーと力を内挿して計算する。ただし Fast_Table を用いた場合、二面角のみの関数となる。

- User Torsion

  ユーザーがソースコードレベルで定義したポテンシャル関数を用いることができる。

2.6.4 非結合相互作用ポテンシャル

Interaction site 間の相互作用ポテンシャル。前述のとおり、**COGNAC** では Atom 以外に Interaction site を定義して、ポテンシャルを計算する。

以下の種類が利用できる。

- 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

  - cutoff 長より長い距離のエネルギー、力を補正する tail correction の機能を持つ。この場合は U_{cutoff} の項は加算されない
  - Tail correction を行わない場合は U_{cutoff} の項を加算し、cutoff 距離で $U_{nonbond} = 0$ になるように自動的にシフトしている

  COGNAC においては 2 原子から定義される Interaction Site にこの Lennard-Jones ポテンシャルを作用させた場合、2 原子の中心が LJ 球の中心になる

- Lennard-Jones with excluded volume [28]

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

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

- General Lennard-Jones

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

  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 [29, 30]

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

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

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

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

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

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

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

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

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

  - 実際のエネルギー計算においては、U_{cutoff} の項を自動的に加算し、cutoff 距離で $U_{nonbond} = 0$ になるようにシフトしている。Tail correction の機能は無い
  - σ_0 はインプットされる値を用い、d_i、d_j からは計算されない仕様になっているので注意

- Gay-Berne - Lennard-Jones pair [30]

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

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

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

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

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

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

  - Gay-Berne - Lennard-Jones pair において、実際の計算は $\alpha = \alpha' = 1.0$ で行う
  - 実際のエネルギー計算において、U_{cutoff} の項を自動的に加算し、cutoff 距離で $U_{nonbond} = 0$ になるようにシフトしている。Tail correction の機能は無い

  - σ_0 はインプットされる値を用い、d_i、d_j からは計算されない仕様になっているので注意

- Table

  Interaction site 間の距離 r_{ij}、あるいは r_{ij} と温度 T の組み合わせを変量とするエネルギー値 $U_{nonbond}$ のテーブルデータにより、任意の距離と NVT アンサンブルなどにおいて設定した温度におけるエネルギーと力を内挿して計算する。ただし Fast_Table を用いた場合距離のみの関数となる。球対称の one atom site のみをサポートする。

  - Tail correction のフラグを off にしておくと、U_{cutoff} の項を加算し、cutoff 距離で $U_{nonbond} = 0$ になるようにシフトする。ただし Tail correction の機能は無いのでフラグ on の場合、補正はされない。

- DPD Groot らの [12, 13] 報告する DPD において、用いられている Nonbonding potential。

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

  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

  ユーザーがソースコードレベルで定義したポテンシャル関数を用いることができる。

Tail correction 機能

非結合間ペアポテンシャルの計算において、一般的にカットオフ距離 r_{cutoff} を設定し、r_{cutoff} 以上の距離のポテンシャルは計算しない。このカットオフにより、無視されたエネルギーと力を補正するために Tail correction が行われる。エネルギーに関しては、補正のあるなしで Dynamics に影響は与えないが、特に定圧条件でシミュレーションを行う場合、力の補正は重要である。

具体的な Tail correction の方法としては、r_{cutoff} より長い距離の動径分布関数 $g(r) \approx 1$ と仮定して 2.79 式で示すようにポテンシャルを解析的に積分してカットオフ距離以遠のペアからのエネルギーを補正する。力に関して、力のベクトルそのものは等方性の仮定によりゼロとなるので、2.80 式により圧力の補正を行う

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

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

N:number of atom, ρ:density

現在は Lennard-Jones および General Lennard-Jones ポテンシャルのみで Tail correction が利用できる

2.6.5 外場

COGNAC では外場によるポテンシャルとして、壁面のポテンシャル、電場等の一様外場など以下に示すようなものを与えることができる。

- Lennard-Jones atomic type potential

  固体壁として、壁面の正方格子上に LJ potential を持つ点を置き、相互作用を計算する。後述の Flat wall に比較して計算時間を必要とするが、壁面と平行方向の力が作用されるので壁のずり等のシミュレーションを行うことができる。

- Lennard-Jones type flat wall [31]

  Lennard-Jones ポテンシャルを壁面方向に積分した、2.81 式で表されるポテンシャルを Interaction site に作用させる。直方体セルのみサポート。

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

σ_{wall}:Lennard-Jones radius, ϵ_{wall}:Lennard-Jones energy, ρ_{wall}:density of wall

図 2.2 に Lennard-Jones atomic type potential と Lennard-Jones type flat wall を比較して示す。

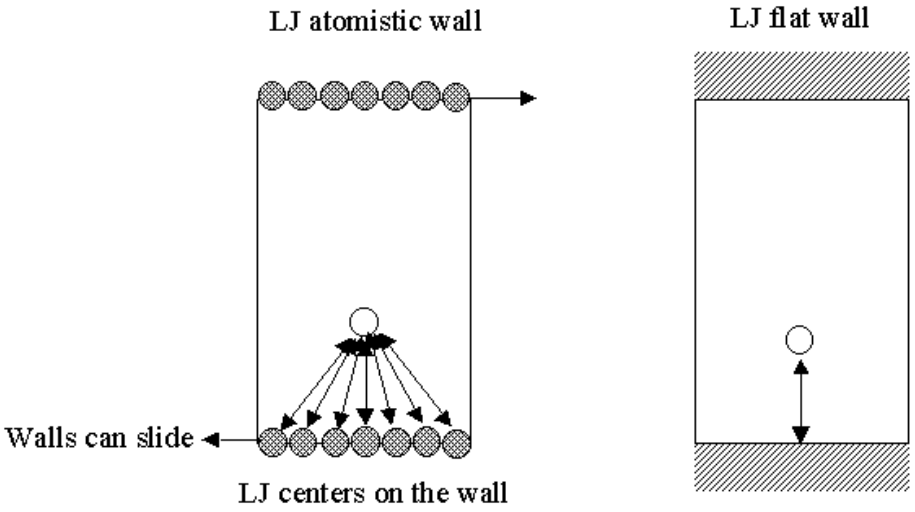


図 2.2: LJ atomic wall と LJ type flat wall の比較

また、上記の 2 種類の固体壁の外場を作用させて場合に、§5.10 で出力する”Wall Pressure”, $\mathbf{P}^{wall}$ は以下の様に定義される。

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

ここで、N は総原子数、A は壁の面積、$\mathbf{f}_i^{wall}$ は固体壁が原子 i に作用する力のベクトルで、Flat wall の場合 $\mathbf{f}_i^{wall}$ は壁面に垂直方向の成分のみを持つが、Atomic wall の場合は壁面に平行方向の成分も持つ。

- Density biased potential [1, 32, 33, 34]

  SUSHI あるいは **Muffin_phaseseparation** により求めた規則格子点上のセグメント種の体積分率の結果を読み込んで、2.83 式に示すように、設定した χ パラメータに応じたポテンシャル $U_{ext}(\mathbf{r})$ を外場として与える。図 2.3 の例では MD の Atom A にかかる外場は 2.83 式で与えられる。ここで $\phi_{A,B}(\mathbf{r})$ は、**SUSHI** あるいは **Muffin_phaseseparation** により求められた、セグメント種 A、B の格子点上の体積分率 $\phi_A(i,j), \phi_B(i,j)$ 等より内挿により求める。

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

  m, n:segment type, χ_{mn}:χ parameter between segment type m and n, $\phi_n(\mathbf{r})$: volume fraction of segment type n at position $\mathbf{r}$

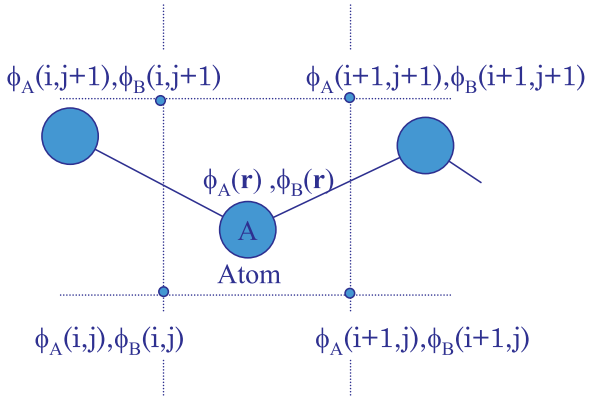


図 2.3: Density Biased potential

COGNAC version 4.2 からは、以下に記す濃度場データを読み込むいくつかのポテンシャルを含めて、**COGNAC** において出力する **Grid_Density** のデータも同様に読み込めるようになっている。たとえば、DPD により求めた濃度場を読み込み、ビーズ－スプリングモデルなど他のモデルにおいて用いるというようなことが可能になっている。

- Lennard Jones type density oriented potential Density biased potential と同様に、**SUSHI** あるいは **Muffin_phaseseparation** により求めた規則格子点上のセグメント種の体積分率に従った、外場を作用させる。ポテシンャル関数としては、2.84 式に示すように、Lennard Jones ポテシンャルに類似した表式を取り、読み込んだ濃度場から内挿により求めた Atom の座標 $\boldsymbol{r}$ 上の濃度 $\phi(\boldsymbol{r})$ を、距離と関連付けている。

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

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

- Reciprocal power type density oriented potential Density biased potential と同様に、**SUSHI** あるいは **Muffin_phaseseparation** により求めた規則格子点上のセグメント種の体積分率に従った、外場を作用させる。ポテシンャル関数としては、2.85 式に示すような表式をとり、斥力のみを作用させる。

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

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

- Total density constrain

2.86 式に示すような、原子の座標から求めた、規則格子点上の全原子種の体積分率に係数をかけたポテシンャルを外場として与える。規則格子点上の体積分率は §2.10 で説明する方法で計算し、そこで得られた規則格子点上の体積分率 ϕ を元にして、上に説明した Density biased potential と同様に $\phi(\mathbf{r})$ を求める。

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

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

- External angle potential

空間領域を指定して、その領域に存在する結合角にのみ cosine harmonic type の結合変角ポテンシャルを作用させる。部分的に配向した構造（結晶ラメラなど）を作成する際に利用できる。

- External torsion potential

空間領域を指定して、その領域に存在する結合二面角にのみ cosine polynomial type の結合二面角ポテンシャルを作用させる。部分的に配向した構造（結晶ラメラなど）を作成する際に利用できる。

図 2.4 に External angle/torsion を作用させたイメージを示す。

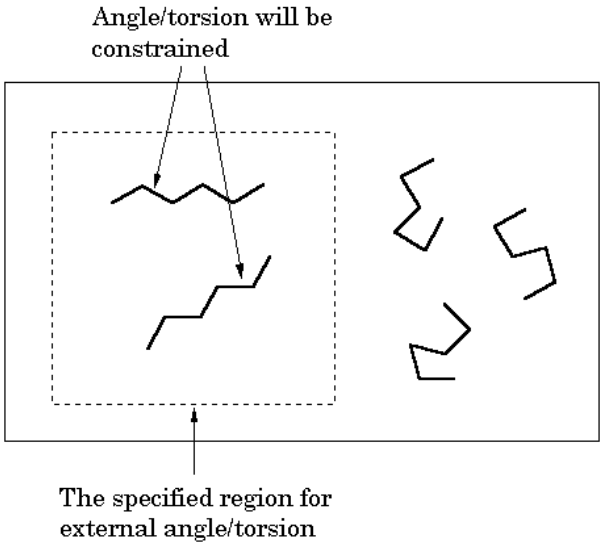


図 2.4: External Angle/Torsion

- Homogeneous field

  3 次元ベクトルで与えられる一様な外場 **V** を Interaction site に作用させる

$$U_{ext} = \mathbf{V} \cdot \mathbf{r} \tag{2.87}$$
$$\frac{dU_{ext}}{d\mathbf{r}} = \mathbf{V} \tag{2.88}$$

- External velocity field

  Muffin_phaseseparation により得られる速度場を、2.89-2.90 式で示される SLLOD アルゴリズム [35] により、原子に作用させる

$$\dot{\mathbf{q}}_i = \frac{\mathbf{p}_i}{m} + \mathbf{q}_i \cdot \nabla \mathbf{u} \tag{2.89}$$
$$\dot{\mathbf{p}}_i = \mathbf{F}_i - \mathbf{p}_i \cdot \nabla \mathbf{u}, \tag{2.90}$$

  u : streaming velocity.

  この場合、**Muffin_phaseseparation** において流れ場を計算する際に用いた境界条件（例えば、Lees-Edwards や固体壁等）に対応する境界条件を **COGNAC** の入力においても指定することが必須である。

- Tethered Force

  指定した **Interaction_Site** を初期座標 $\mathbf{r_0}$ を中心として、2.91 式に従い拘束する

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

- User External Potential

  ユーザーがソースコードレベルで定義したポテンシャル関数を用いることができる。

2.6.6 静電相互作用ポテンシャル

Interaction Site 間の点電荷－点電荷間あるいは双極子－双極子間の静電相互作用ポテンシャルを計算する。方法として以下のものがある。

- Cutoff 法

  通常の非結合間相互作用と同様、カットオフ距離を設定し、それより遠い相互作用は計算しない。点電荷間のクーロン力の計算は、2.92 式に従う

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

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

- Cutoff with Debye function

  カットオフによるクーロン相互作用計算の際、2.93 式のようなダンピング関数を付加する

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

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

- Reaction Field[36, 37, 38]

  双極子および点電荷間の静電相互作用の計算において Reaction field 法による長距離の補正をサポート

- Ewald[4]

  Ewald 法による点電荷間静電相互作用の計算

  Ewald パラメータの自動設定
  Ewald 法の波数空間におけるパラメータ設定を自動 (**AUTO**) にした場合、以下の式を用いて値が設定される

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

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

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

- Field electrostatic[39]

  DPD に付加して静電相互作用を計算する方法。Atom の持つ電荷より格子点上に電荷を割り振り、Poisson 方程式を解くことにより電場を計算する。

- Particle-Particle Particle-Mesh(PPPM) 法 [40, 41]

  PPPM 法による点電荷間静電相互作用の計算

2.7 初期構造作成

入力 UDF で定義された分子アーキテクチャーに対応して、実際のシミュレーションを行うための初期座標を作成する方法を以下に解説する。

2.7.1 初期座標発生

- Random : 非晶構造を生成する

  結合長ポテンシャルで与えられる平衡結合長に基づいて、ランダムに末端の原子から座標を発生して分子鎖をのばしていく。この際の、結合角を結合角ポテンシャルで与えられる平衡角に固定するオプション、および二面角の状態（トランスーシスの two state あるいはトランスーゴーシュの three state）を温度とコンフォメーション間のエネルギー差を指定することにより、制御するオプションを選択することが出来る。
  なお、初期構造を発生する際には排除体積は考慮していないので、通常後述の構造緩和を行う必要がある。

- Helix：規則的に配置されたらせん構造を生成する

  分子配置の原点として単純立方格子、面心立方格子、体心立方格子、あるいはユニットセル内でのシフト値を指定して配置することが出来る。指定された原点から、繰り返される二面角の配列を指定して一方方向に分子鎖をのばし、らせん構造を作成する。この際、主鎖に不斉中心を持つ場合は meso-racemi の指定が可能。

- Crystal：結晶構造を生成する

  格子定数、対称操作、fractional coordinate 等を UDF から読み込み、任意の結晶構造を作成する

- Lamella：半結晶ラメラを生成する

  平均場理論により予測される結晶ラメラ間の非晶部の分子鎖の構造を再現するように、結晶ラメラの初期構造を生成する。
  アルゴリズムの詳細に関しては次節で解説する。

- **SUSHI** の出力を用いた多相構造を生成する

  SUSHI によって計算された、ブロックコポリマーやポリマーブレンドのモルフォロジー（セグメント種およびセグメント濃度分布）を再現した分子鎖の初期座標を生成する機能を、非晶構造作成機能の拡張として持つ。
  アルゴリズムの詳細に関しては以下の節で解説する。

Density biased Monte Carlo

土井プロジェクトにおいて、**SUSHI** により計算されるセグメント濃度分布より、分子鎖の初期構造を効率的に生成する方法を開発した [1, 34, 42]。以下にその方法を解説する。

SUSHI で行っている経路積分を用いた平均場理論より、個々のセグメントの分布、体積分率を求めることが出来る。例えばセグメント重合度 N の分子鎖において、個々のセグメントの体積分率は $\phi_n(\mathbf{r})$ で与えられる。ここで n はセグメント ID(=1,2,⋯N)、$\mathbf{r}$ は位置である。ただし、格子モデルによる平均場計算では、位置 $\mathbf{r}$ は 3 次元の場合、格子点 (i,j,k) 上の離散的な位置におけるセグメント体積分率 $\phi_n(i,j,k)$ を与える。

MD シミュレーションに用いる分子鎖の初期座標を発生させる際に、この平均場理論で求められるセグメント体積分率 $\phi_n(i,j,k)$ により重みを付けたモンテカルロ法により、順次鎖を成長させていく。重合度 N の単分散の系で、平均場計算のセグメントと MD で用いるセグメント（ビーズ）が 1 対 1 対応をしている場合を例に取り手順を以下に示す。

1. n=0（末端）のセグメントの座標を決定
   乱数により発生した、座標 $\mathbf{r}$ におけるセグメントの存在確率を体積分率 $\phi_0(\mathbf{r})$ とおく。$\phi_0(\mathbf{r})$ は周囲の格子点上のセグメント体積分率 $\phi_n(i,j,k)$ から内挿して求め、[0,1] の一様乱数とを比較して、その座標の採択を判定する。

2. 分子鎖を順次成長させる

   図 2.5 に示すように ID=$n-1$ のセグメントが $\mathbf{r}'$ に存在するとき、セグメント n は図に示す球面 $\mathbf{r}+\boldsymbol{\ell}$ 上に存在する。ここで $|\boldsymbol{\ell}|=$ b、ただし b は平衡結合長で、初期座標生成の際は固定する。乱数により $|\boldsymbol{\ell}|=$ b を満たす $\boldsymbol{\ell}$ を発生し、周囲の格子点上の体積分率 $\phi_n(i,j,k)$ から内挿により求められる $\phi_n(\mathbf{r})$ に基づいて $\boldsymbol{\ell}$ を選択する。この場合格子数あるいは重合度が非常に大きくセグメント単位の体積分率の絶

対値が小さすぎるような場合は、サンプリングの効率を上げるため、あらかじめ 10-100 回程度 $\boldsymbol{\ell}$ をランダムに生成しサンプリングを行い、$\phi_n(\mathbf{r}+\boldsymbol{\ell})$ の最大値 $\phi_{n,max}(\mathbf{r}+\boldsymbol{\ell})$ を見積り、実際の座標決定の際は、新たに発生させた $\boldsymbol{\ell}'$ より $\phi_n(\mathbf{r}+\boldsymbol{\ell}')/\phi_{n,max}(\mathbf{r}+\boldsymbol{\ell})$ と [0,1] の一様乱数を比較して採択を決定する。

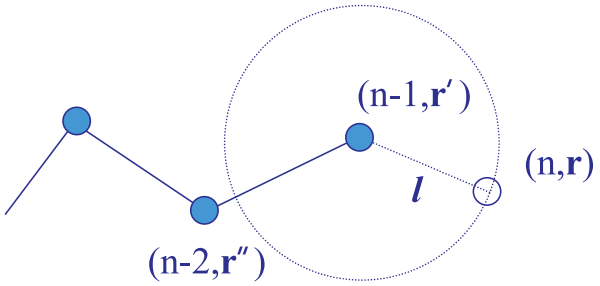


図 2.5: Density biased Monte Carlo

Lamella generator

土井プロジェクトにおいて、平均場理論により求められる結晶間非晶（インターラメラ）におけるループ鎖／ブリッジ鎖の確率に基づき、分子鎖の初期構造を精度よく効率的に発生させる手法を開発した。[34]

以下にその手法の概要と結果を示す。

結晶-非晶ラメラ初期構造構築の方法

1. 構築する結晶-非晶ラメラの長周期、結晶相の長さを指定する
2. 結晶及び非晶相の密度を指定する
3. 入力で指定される分子鎖の座標を末端から順次生成する。この際、排除体積効果は考慮しない。またこの場合、開始末端は必ず非晶領域におく
4. 分子鎖末端からは結晶領域に入るまで、非晶内をランダムウォークして座標を生成する
5. 生成する原子が結晶領域に入ったら、結晶構造に対応するらせん構造に従い、非晶領域に末端が届くまでのばす
6. 生成する原子が非晶領域に入ったら、平均場により得られたループ鎖／ブリッジ鎖の分布に基づいて、非晶層部分の座標を非晶の末端まで生成する
7. 鎖末端までを 5-6 を繰り返す。この際末端が結晶領域で終わる場合はステップ前の非晶鎖の生成よりやり直し、末端が非晶部分に存在するようにする
8. 設定の分子数、ステップ 3-7 を繰り返す
9. すべての分子の初期座標より結晶部の密度を計算し、設定密度と大きくずれる場合は、すべての分子の座標生成をステップ 3 からやりなおす
10. 初期座標が生成されたら、排除体積効果を導入して構造緩和を行う

この手順の中で本プロジェクトにおいて独自に開発された Step 6 の部分の詳細を以下に解説する。

まず、平均場計算から求められる、ループ鎖／ブリッジ鎖を合計した非晶鎖全体の密度 d_a を 2.96 式より計算する。

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

ここで、$n_l(m)$、$n_b(m)$ はそれぞれ平均場計算から求められる結晶鎖一本あたりの重合度 m のループ鎖、あるいはブリッジ鎖の存在確率、また L_a は非晶部分の厚みである。テール鎖および結晶に取り込まれない自由鎖の存在を近似的に無視すれば、結晶密度により規格化された d_a は 1.0 になる。

実際には結晶相／非晶相の密度を別途設定できるので、与えられた非晶密度／結晶密度比 r_d より、1 本の分子鎖が結晶相より非晶相に進行していく確率 p_a と 非晶相に入らず折り返して結晶相に戻る（タイトフォールディング）確率 p_f はそれぞれ 2.97、2.98 式で与えられる

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

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

2.97 式の総和の部分は、テール鎖を無視した場合、平均場により求められる結晶一本あたりの非晶の本数で $r_d = 1$ の場合の p_a に相当する。

ただし、本方法においては、長さ 1 のループ鎖とタイトフォールディングの区別を付けることをしておらず、タイトフォールディングでも必ず長さ 1 の非晶部分を持つとする。この場合、全非晶相の密度 $d'_a(= r_d)$ は 2.99 式で与えられる。

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

ここで、右辺第一項は平均場に基づいた非晶部分由来の項、第二項はタイトフォールディング由来の項である。

2.99 式より p_a、p_f は 2.100、2.101 式のように導かれる。

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

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

実際に構造生成の過程において、分子鎖が結晶相の末端まで伸びた場合、平均場計算で得られる $n_l(m)$、$n_b(m)$ に基づきモンテカルロ法により次に非晶鎖として発生させる分子構造（ループ鎖／ブリッジ鎖および重合度 m）を選択する。ただしその際、タイトフォールディングを考慮するために、$n_l(1)$ の時のみ、平均場で得られる $n_l(1)$ に 2.100、2.101 式で得られる p_f を加えた値を確率分布として用いる。

そして、選択された構造に従いランダムに鎖の座標を生成し、選択された長さ m で目的のループまたはブリッジの構造を取るまで試行を繰り返す。

2.7.2 構造緩和機能

発生させた初期座標より半自動的（ポテンシャルスケーリング等を自動的に行う）に緩和構造を作成できる。Dynamics と Minimize による構造緩和を行うことができる。

Dynamics による構造緩和

MD を用いて構造緩和を行う場合、設定したポテンシャル関数をそのまま適用すると排除体積効果により、きわめて大きな力が作用し、正常に計算することができない。そのため **COGNAC** においては、力をスケールすることにより排除体積を徐々に導入するという方法をとる。具体的には atom にかかる最大の力の許容値を設定し、それ以上に力がかかれば、最大値にスケールするという方法をとる。そして、緩和の過程で、個々の atom にかかる力の平均、最大値が低下していくと、スケールする maximum force を徐々に大きくしていき、最終的にすべての atom にかかる力が初期設定の maximum force 以下になると緩和終了とする。

Minimize による構造緩和

緩和の方法として Minimize を選択した場合、最急降下法と共役勾配法を順次適用してエネルギー極小化を行う。

2.8 境界条件

シミュレーションを行う際の境界条件として以下の条件をサポートする。

- 2 次元および 3 次元周期境界条件
  立方体セル、直方体セル、斜方セルに対応
- Lees-Edwards 境界条件 [43]
  せん断流動を与える場合の境界条件をサポート
- スタガード反射境界条件
  土井プロジェクトにおいて新規に開発された、反射境界条件 [34, 44] をサポート
- 周期境界条件に基づいた化学結合（Periodic chain）
  定義されている bond、angle、torsion を、与えられた周期境界条件に基づいて、最短の距離になるイメージの atom 間から計算する機能を持つ。無限長鎖のモデリング等に用いる。

スタガード反射境界条件

土井プロジェクトにおいて開発された、新たな境界条件、スタガード反射境界条件 (Staggered reflective boundary condition) に関して解説する。図 2.6(a) に示す様に通常の反射境界条件では、ユニットセルから出た原子は境界面に対称な位置にイメージを持つ。高分子鎖の場合、このようなイメージを考えると結合している実在原子とのオーバーラップが生じ、正常に計算を行うことが出来ない。そこで、図 2.6(b) の様に、境界面に平行に 1/2 セルサイズ分ずらした位置にイメージをおくことにより MD シミュレーションにおいて反射境界条件を実現する。

2.9 化学反応

COGNAC では化学反応を模した、粗視化モデルにおける結合の生成、切断、重合反応、および Atom Type の置換を考慮した分子動力学シミュレーションを行うことが可能である。以下にその方法論を解説する。

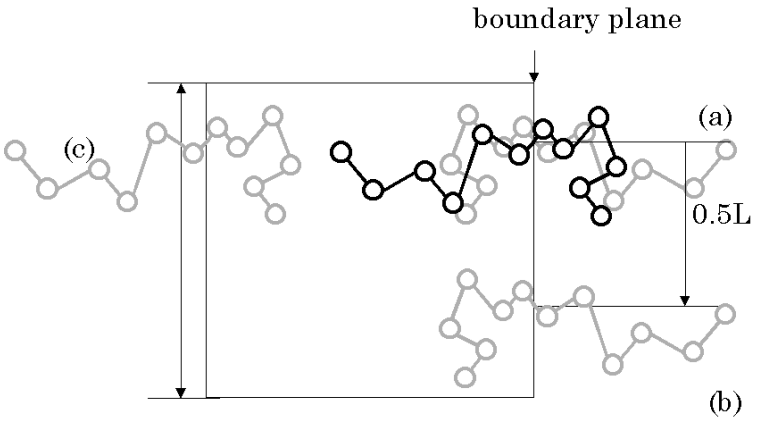


図 2.6: スタガード反射境界条件: 黒線は分子の実座標の表示、灰色の線は各境界条件におけるイメージ (a) 反射境界条件、(b) スタガード反射境界条件、(c) 周期境界条件

2.9.1 結合の生成、重合反応

COGNAC においては、架橋、縮重合反応のように $A+B \rightarrow A-B$ という原子種の変更のない単純な結合の生成に加えてラジカル、イオン反応のように $R+M \rightarrow P-R$（ラジカル活性末端 R がモノマー M と反応し不活性ポリマーユニット P になり、新たにモノマー M が活性末端 R になるというようなイメージ）というような反応を扱うことが出来る。

結合の生成は、入力において設定した条件にしたがって、MD の過程で結合生成のチェックを行う。以下にその手順を上げる。

1. 入力における設定項目

   - 結合生成の可能性のある Atom Type、その原子種が持ちうる最大結合数を指定する
   - 実際に生成しうる結合をなす Atom Type のペアを先の AtomType の指定の中から選択する
   - 結合判定のチェックを行うインターバルを設定する
   - 結合生成の閾値となる距離を指定する
   - 結合生成の確率を指定する
   - 分子内結合を許可するかどうかのフラグを設定する
   - 生成した結合の Bond Potential Name を指定する
   - 生成した結合により新たに定義される結合角、結合二面角の Potential Name を指定する（オプション）

   重合反応の場合、上記に加えて以下のような設定を行う

   - 生成する活性末端の原子種
   - 生成する不活性原子の原子種
   - 生成する活性末端の原子名（オプション）
   - 生成する不活性原子の原子名（オプション）

2. MD シミュレーションにおける結合生成のチェック

   入力で設定したインターバル (time step) 毎に、結合可能な atom 間の距離を計算し、指定した閾値以内に近づいている場合、指定された反応確率に基づいて結合を生成する。結合が生成する場合は atom 間の

Pair interaction を off にし、Bond potential を on にする。ただし、分子内結合を許可していない場合、分子内の atom ペアが近づいても結合は生成しない。
反応性は基本的に反応確率で指定するが、閾値距離あるいは反応チェックのインターバルを変えることによっても差をつけることができる。

3. トラジェクトリの出力

結合が生成して新たに同一分子となった場合、トラジェクトリの **Set_of_molecules**、**Structure** の構造もそのように変更される。そのため、同一の atom でも mol index、atom index が変化するので注意。唯一 Atom_ID は、いかなる場合でも変化しない。

2.9.2 Atom Type の置換

COGNAC は、入力において設定した条件にしたがって、MD の過程で Atom Type の変更する機能を持つ。この機能は、擬似的な溶媒蒸発（溶媒 Atom が気相に移動した際、Void atom に変換する）などのシミュレーションに有効な機能である。以下にその手順を上げる。

1. 入力における設定項目

   - 変更の対象となる Atom Type Name
   - 変更後の Atom Type Name
   - 変更後の Atom Name （オプション）
   - 変更後の Interaction Site Type Name
   - Atom Type 変更のチェックを行うインターバルを設定する
   - Atom Type 変更の確率を指定する
   - Atom Type 変更の判断タイプを指定する。現バージョンでは空間領域を指定する機能のみ

2. MD シミュレーションにおける原子種の変更

   入力で設定したインターバル (time step) 毎に、変更対象の Atom type を持つ atom の位置をチェックし、指定した領域に存在する場合、指定した確率に基づいて、Atom Type,Atom Name(option),Interaction Type の変更を行う。

3. トラジェクトリの出力

   Atom Type が変更された場合、トラジェクトリに **Set_of_molecules** も出力される。

2.9.3 結合の解離

結合の解離は、入力において設定した条件にしたがって、MD の過程で結合解離のチェックを行う。以下に結合解離の条件とその手順を上げる。

結合解離の条件

1. 結合長による結合解離判定

   通常の熱分解反応等を模擬して、結合長が閾値を超えると結合を解離する。

2. 位置による結合解離判定

   結合の両端の Atom のどちらかが、指定した領域に存在する場合、結合を解離する。低分子の擬似的な蒸発を取り扱う場合、蒸発した低分子の結合を消去し Void 粒子に変換する場合に用いることが出来る

結合解離の手順

1. 入力における設定項目

   Bond_Potential 毎の解離条件設定

2. MD シミュレーションにおける結合解離のチェック

   Bond potential が設定されている結合をポテンシャル計算毎に、条件に従い判定し、条件を満たせば結合を解離して、関与する bond potential、angle potential,torsion potential を off にし、pair interaction を on にする

3. トラジェクトリの出力

   結合の生成と同様に **Set_of_molecules**、**Structure** の構造が変化する。

これら結合の生成、解離、Atom Type の置換は同時に設定することが可能でシミュレーション過程で生成した結合が解離する、あるいはその逆も可能である。

2.10 濃度分布計算法

COGNAC においては、**SUSHI** 等の連続体モデルと濃度分布などの計算結果を比較するために、atom の座標データより、規則格子点上の濃度あるいは体積分率を近似的に求める機能を持つ。以下にその方法を解説する。**COGNAC** エンジン出力においては方法 2 で紹介する外挿法を採用している。ただし、第 7 章で紹介する Python script による 1 次元方向の濃度分布の解析は上の単純分割法により計算している。

方法 1: 単純な領域分割

図 2.7 に示すようにユニットセルを分割し、その分割した単位セル内に存在する Atom の数を数えあげて、単位セルの体積で割ることにより、単位セルの密度を計算する。

1 軸方向の濃度分布の様に、Atom 数に比較して単位セル数を小さく取って、同一単位セル内に存在する Atom 数が多くなるように場合では、比較的精度よく濃度分布を計算することができる。

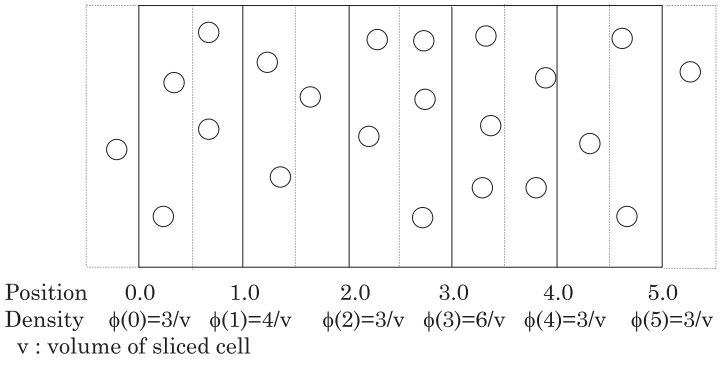


図 2.7: 単純な領域分割による濃度分布

方法 2: 外挿法

3 次元に単位セルを分割し、単位セル数と Atom 数が同程度になると、単純な領域分割による Atom の数え上げでは、精度よく濃度分布を求めることができない。そこで Atom の体積を考慮し、1 つの Atom の存在を周囲の複数の格子点上の濃度分布として計算する方法により、格子点上の濃度分布をいくらかスムースに変化させる。

この方法の詳細を 1 次元の場合を例にとり図 2.8 に示す。図において、1 次元上の $-1.0, 0.0, 1.0, 2.0$ という格子点を定義し、atom が 0.4 の位置に存在するとする。図 2.8(a) に示すように Atom の有効半径を 1.0 とした場合、三角形で示されるように $0.0, 1.0$ の 2 格子点上に濃度分布を外挿法により Atom 由来の存在確率を与える。図 2.8(b) においては $-1.0, 0.0, 1.0, 2.0$ の 4 点に Atom の存在確率を与える。一般的に、Atom が z_{atom} にあり、有効半径 r とした場合、各格子点 z の存在確率比 $\phi(z)$ は 2.102 式で求められる。

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

実際には $\sum_z \phi(z) = 1.0$ になるように規格化した値を格子点上の 1Atom 由来の存在確率とし、対象となるすべての Atom に関して足し合わせた値が格子点上の Atom 数になり、これを単位セル体積で割ることにより濃度を求める。

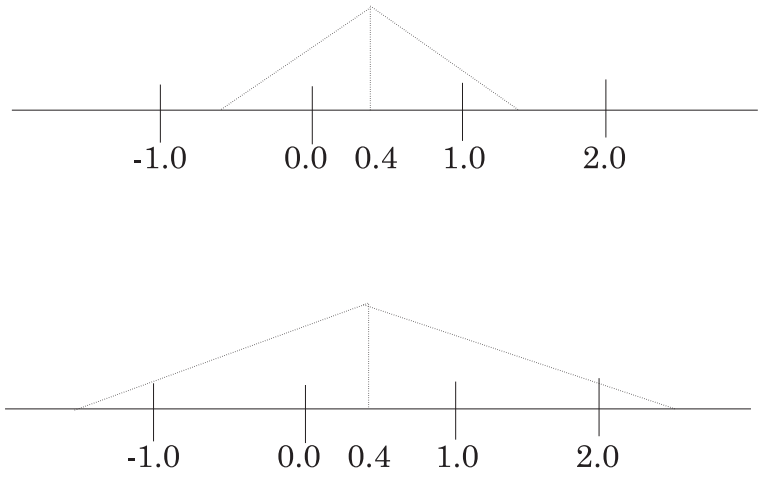


図 2.8: 外挿法による濃度分布 (a)r=1.0 の場合, (b)r=2.0 の場合

3 次元の場合、格子点の座標を $\mathbf{r}(x, y, z)$ とした場合各々1 次元で計算した、存在確率 $\phi(x)$、$\phi(y)$、$\phi(z)$ を掛け合わせることにより、2.103 式のようにして、求めることができる。

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

2.11 On the fly 自己相関関数計算

分子動力学シミュレーション結果の解析として、座標、速度、結合ベクトルあるいは応力、エネルギーなどの物理諸量の経時データの自己関数を計算することは、定常的に行われる。任意の物理量 A の規格化された自

己相関関数 $C(t)$ は 2.104 式で表され、実際の計算には時間 t だけ離れた 2 つのデータの多数の組み合わせを用いる。

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

通常の解析では、座標、物理量のデータをファイルに保存し、シミュレーション後に解析を行うが、全てのタイムステップのデータを保存するにはデータが膨大になり、また処理時間もかかるため、一定の間隔でデータを保存する。そのため使用されるデータ点数が少なくなるため、自己相関関数の精度は低下する。

そこで **COGNAC** では、Magatti らのアルゴリズム [45] を用いて、経時データの自己相関関数をデータをファイルに保存することなく、シミュレーションと同時に (on the fly) 解析する機能を持たせた。この Magatti らのアルゴリズムでは、自己相関は指数関数的に減少することを前提とし、短時間間隔の変化は、一ステップのデータを用い、長時間間隔では、数ステップの平均値間の相関をとる。この平均を取るステップ数と、時間間隔を数段階に分割することにより、少ないメモリーで長時間の相関関数が計算できる。

バージョン 8.3 においては、応力の自己相関関数のみが実装されている。応力の自己相関関数より 2.105 式の Green-Kubo 公式より、応力緩和 $G(t)$ を得ることができる。

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

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

第3章　操作入門

ここでは、ビーズ−スプリングモデルによる ABA トリブロックコポリマーのシミュレーションを例にとり、ブロックコポリマーの構築法、ビーズ−スプリングモデルの計算条件設定などに関して説明する。
使用するファイルは特に指定するものを除いて、“sample/block” 以下にある。ディレクトリパスが指定されている場合は”COGNAC” ディレクトリからの相対パスを示している。

3.1　Action SILK による入力データ作成

ここでは、**Action SILK** を用いて ABA トリブロックコポリマー 50 分子のトポロジー情報を作成する手順について説明する。

1. **SILK** 入力ファイル（“potential_map.udf”）

   まず最初に **SILK** 入力ファイル（“python/silk/sample/potential_map.udf”）を参照する。ここでは、5 番目のレコード（Record Number は **4**、Record Label は “BS_ABA_triblock”）を使用する。

2. UDF（“potential_map.udf”）の Open

   GOURMET を起動し、**File** → **Open...** メニューから UDF（“potential_map.udf”）を Open する。画面には Record Number **0** のデータが表示されているので、**GOURMET** 窓下のつまみを右にスライドさせ、右下の窓に Record Label である “BS_ABA_triblock” が表示されるまで動かす。

3. 分子の作成

   (a) **Action SILK** ポリマーテンプレートコマンドの選択

   読み込んでいる “potential_map.udf” の **Set_of_Molecules** のフォルダを右クリックすると、図 3.1 のように、いくつかの **Action SILK** コマンドが現れる（初期状態では **Set_of_Molecules** にはデータが含まれていないので、グレーで表示されている）。この中より、ビーズスプリングモデルのトリブロックコポリマーを作成するコマンド、**SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...** を選択する。

   (b) **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...** の設定

   図 3.1 で **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...** を選択すると図 3.2 に示すようなウインドウが現れる。ここで、トリブロックコポリマーのアーキテクチャおよびポテンシャルタイプ等を設定する。

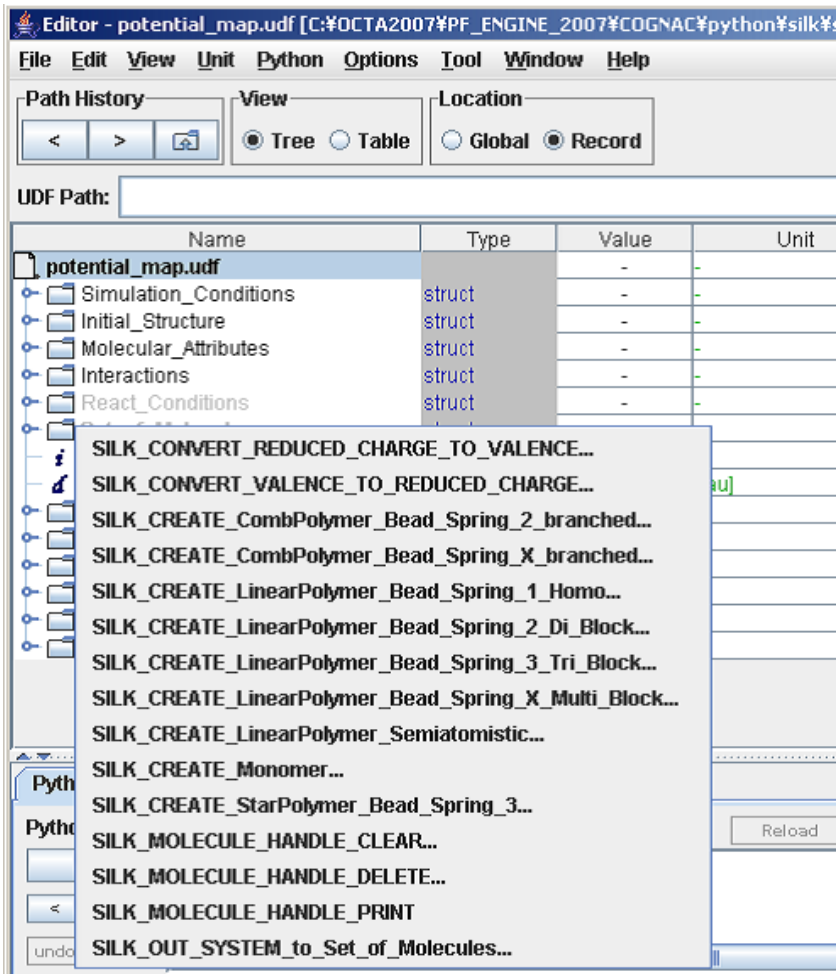


図 3.1: **Action SILK** コマンド起動の例

ここでは分子 A20B40B20 を 50 分子作成する。分子のアーキテクチャー以外に Atom のタイプ、bond potential のタイプ、interaction site のタイプ等もここで指定する。

パラメータの説明

- **name**：分子名、任意の名前でいい。たとえば **A20B40A20**。
- **numMol**：分子数。ここでは **50**
- **atom#_name**：#番目のブロックの Atom name。
  任意の名前でいいがここでは **A**(#=0,2) と **B**(#=1) にする。
- **atom#_num**：#番目のブロックのブロック長。
  ここでは **20**(#=0,2) と **40**(#=1)。
- **atom#_type**：#番目のブロックの Atom の Type。
  “potential_map” の **Molecular_Attribute** に含まれるものから選択する。ここでは、**atom1**(#=0,2)

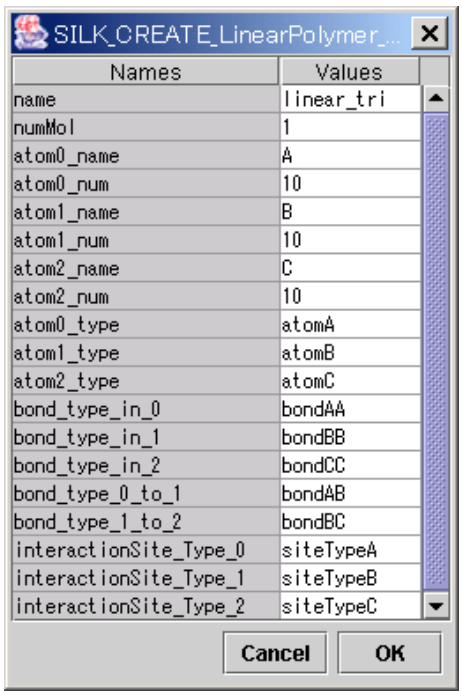


図 3.2: **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...** の初期パラメータ

と **atom2**(#=1)

（注意） Atom name は atom の名前であり、任意に付けることが出来る。一方、Atom type は Atom の種類 (Atomistic MD なら元素種に相当) を意味し、**Molecular_Attribute** に設定されている必要がある。

- **bond_type_in_#**：#番目のブロック内の Bond potential type。
  “potential_map” の **Molecular_Attribute** に含まれるものから選択する。ここでは、bond1(#=0,2) と bond2(#=1)。

- **bond_type_#1_to_#2**：#1 番目と#2 番目のブロックをつなぐ Bond potential type。
  “potential_map” の **Molecular_Attribute** に含まれるものから選択する。ここでは、**0_to_1**, **1_to_2** 両方とも **bond3**

- **interactionSite_Type_#**：#番目のブロックの interaction site type。
  “potential_map” の **Molecular_Attribute** に含まれるものから選択する。
  ここでは、**siteType1**(#=0,2) と **siteType2**(#=1)。

図 3.3 に示すように、すべてのパラメータの入力が終わった後、**OK** をクリック。この時点ではまだ、**Set_of_Molecules** には書き込まれない。

4. **Set_of_Molecules** への書き込み

このサンプルでは一種類のブロックコポリマーを作成するので、ここで定義した、分子を **Set_of_Molecules**

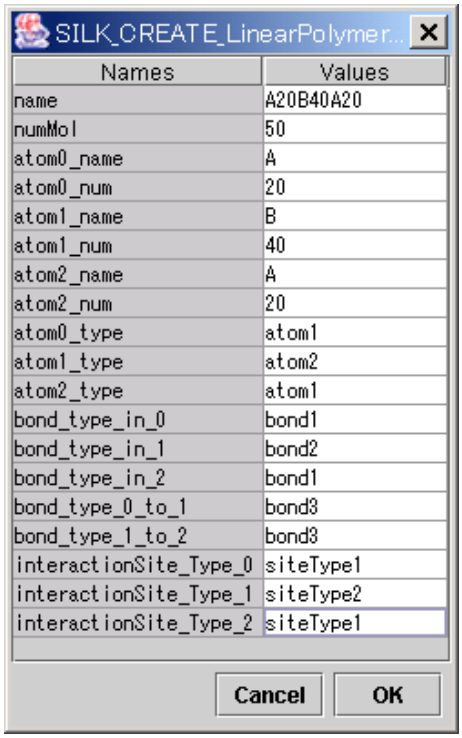


図 3.3: **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block...** の設定パラメータ

へ書き込む。ブレンドを作成する場合は、前項の操作をポリマー種の数だけ繰り返す。

再び、**Set_of_Molecules** のフォルダの右クリックにより現れるコマンドより、**SILK_OUT_SYSTEM_to_Set_of_Molecules...** を選択し現れるウインドウで **Continue** を選択したまま **OK** をクリックすると “potential_map” の該当の Record に **Set_of_Molecules** が書き出される。ただし、この時点では **GOURMET** のメモリキャッシュ内に書き込まれるのみなので、Save しない限り”potential_map.udf” ファイルは更新されない。通常、雛型の”potential_map.udf” ファイル自体には **Set_of_Molecules** を追加しないほうがいい。

5. 新規の UDF ファイルの作成

“potential_map” に書き込まれた **Set_of_Molecules**、および雛型の計算条件を **COGNAC** の入力 UDF として用いるため、新規な UDF ファイルを作成して、そこに “potential_map” の該当の Record を書き出す。
そのために今度は **Editor** ウインドウに現れる “potential_map.udf” のルートのアイコンを右クリックして、現れるコマンド群より **SILK_UDF_CREATE...** を選択する。そこで、現れるウインドウで出力 UDF ファイル名を指定することにより、**COGNAC** 入力 UDF が作成される。ここではあとの説明のため、ファイル名を “A20B40A20_in.udf” としておく。

3.2 入力 UDF の編集と COGNAC の起動

次に、**Action SILK** で作成した **COGNAC** の入力 UDF である “A20B40A20_in.udf” を **GOURMET** で読み込み、計算条件等を編集した後、**COGNAC** を起動する。

1. 入力 UDF の読み込み

GOURMET Editor ウインドウより、**File** → **Open...** を選択して、現れるファイルブラウザから “A20B40A20_in.udf” を選択して読み込む。**GOURMET** ウインドウ左側に、図 3.4 に示すような UDF の構造に応じたツリーが表示される。

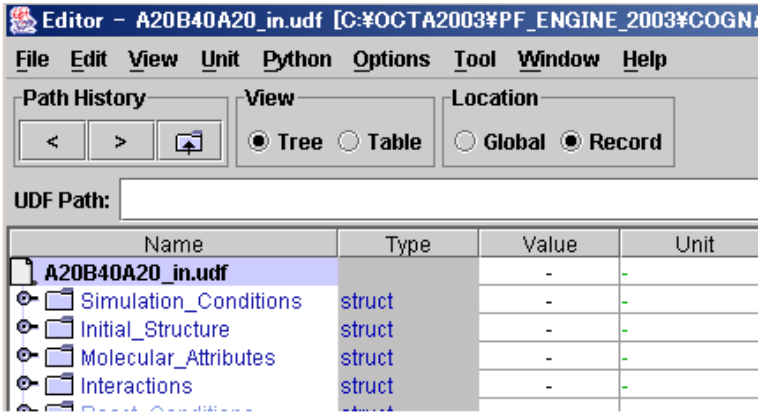


図 3.4: **GOURMET** における UDF ツリー表示の例

あるいは、**View** 選択で **Table** を選択すると図 3.5 に示すような形式で表示される。以下本マニュアルでは特に断りのない場合、Table モードでの表示を例に取り解説する。

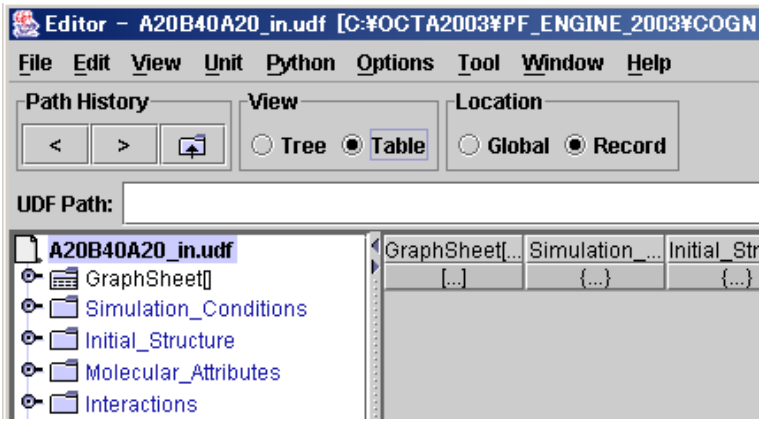


図 3.5: **GOURMET** における UDF ツリー表示 (table mode) の例

COGNAC では入力データはすべて **Global** データとして扱われる。左側に表示されているツリー構造の、フォルダー様のアイコンをクリックすることにより、下層のデータが表示される。
例えば、**Simulation_Conditions** → **Dynamics_Conditions** → **Time** とクリックしていくと、図 3.6 に示すようにタイムステップに関する、入力データが表示される。

2. 初期構造の設定

UDF Path 名 **Initial_Structure** で初期構造に関する情報を設定する。**Initial_Structure** 以下のデー

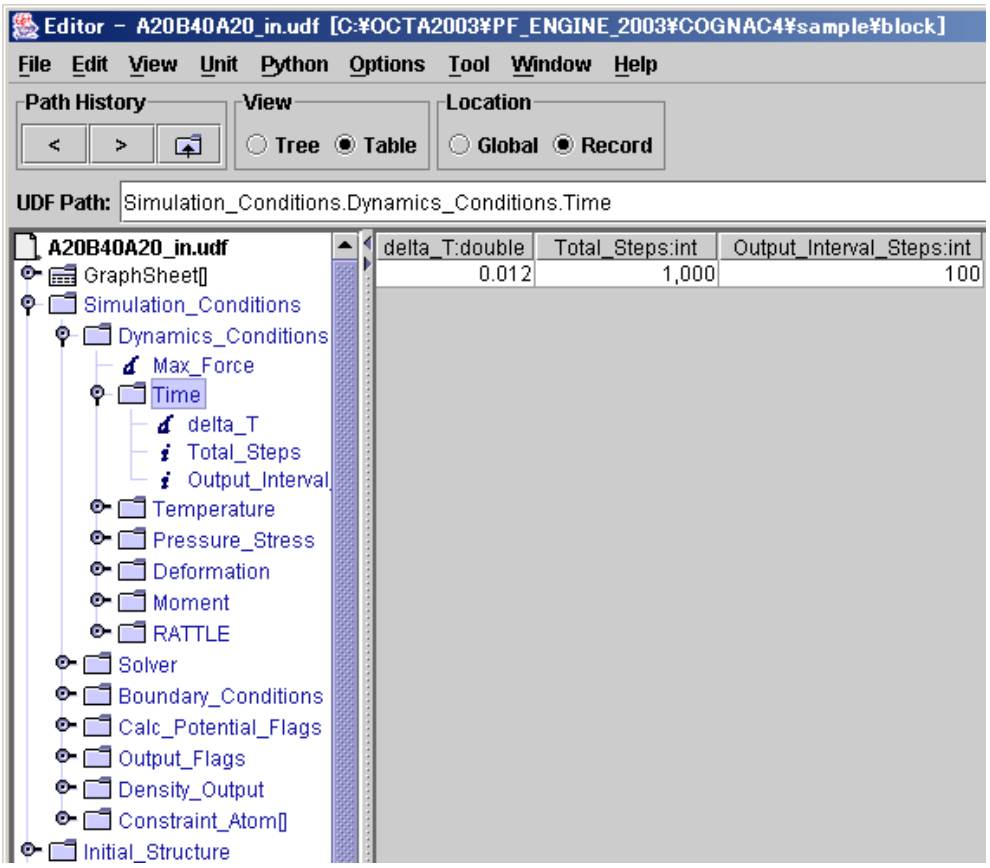


図 3.6: **Simulation_Conditions.Dynamics_Conditions.Time** 表示の例

タの主なものを説明する。

Initial_Unit_Cell.Density··· 初期条件として設定される系の密度である。

ここでは無次元化された密度 **0.85** を入力している。

Generate_Method.Method··· 初期構造作成についての方法を記述。

Value の欄を左クリックすると、セレクトメニューが現れる。その中より **Random** を選択することにより、UDF Path 名 **Generate_Method.Random** 以下のデータが **COGNAC** によって参照される。

同じ階層に存在するその他のデータ（例えば UDF Path 名 **Initial_Structure.Generate_Method.Helix**）は無視されることに注意。

3. 計算条件の設定

UDF Path 名 **Simulation_Conditions** を編集することにより計算条件を設定する。設定したパラメータは以下のとおりである。

(a) ダイナミクス条件

UDF Path 名 **Simulation_Conditions.Dynamics_Conditions** 以下のデータ値について説明する。

Time.delta_T（設定値:**0.012**）･･･ 1 ステップ当りの時間を示す。

Time.Total_Steps（設定値:**1000**）･･･ シミュレーションの総ステップ数である。

Time.Output_Interval_Steps（設定値:**100**）･･･ 出力のインターバルを示す。

Temperature.Temperature（設定値:**1.0**）･･･ 設定温度。

Temperature.Interval_of_Scale_Temp（設定値:**0**）･･･ 速度スケーリングのインターバルステップ数。

*__Temperature.Interval_of_Scale_Temp__ を **0** にした場合、速度スケーリングは実行されない。ここでは別の手法によって温度制御するので速度スケーリングの必要は無い。

(b) ソルバー

UDF Path 名 **Simulation_Conditions.Solver** 以下のデータ値について説明する。

Solver_Type（設定値:**Dynamics**）･･･ ダイナミクス（**Dynamics**）、ミニマイズ（**Minimize**）のいずれかが選択出来る。ここでは **Dynamics** を選択。

Dynamics.Dynamics_Algorithm（設定値:**NVT_Kremer_Grest**）･･･ 実行するアルゴリズム。

Dynamics.NVT_Kremer_Grest.Friction（設定値:**0.5**）･･･ ダイナミクス（アルゴリズム名：**NVT_Kremer_Grest**）を実行するにあたり必要なパラメータ。

(c) 境界条件

UDF Path 名 **Simulation_Conditions.Boundary_Conditions** 以下のデータ値について説明する。

Boundary_Conditions（設定値:**PERIODIC**（**a_axis**、**b_axis**、**c_axis** 全て））･･･ 境界条件の設定

(d) 計算するポテンシャル項の設定

UDF Path 名 **Simulation_Conditions.Calc_Potential_Flags** 以下の主なデータ値について説明する。

Calc_Potential_Flags.Bond（設定値：**1**）

Calc_Potential_Flags.Angle（設定値：**0**）

Calc_Potential_Flags.Torsion（設定値：**0**）･･･ 結合相互作用の計算条件。ビーズスプリングモデルの場合、結合変角ポテンシャルの計算に関する項目:**Angle**、および結合二面角ポテンシャルの

計算に関する項目:**Torsion** は、計算を行わない。したがって設定値を 0 にする。

Calc_Potential_Flags.Non_Bonding（設定値：**1**）· · · 非結合間相互作用の計算条件。

Calc_Potential_Flags.Non_Bonding_1_4（設定値：**0**）· · · 結合二面角ポテンシャルを計算する場合、1-4 結合ビーズ間の非結合間相互作用を計算するかどうかのフラグ。ビーズスプリングモデルの場合、結合二面角ポテンシャルは計算しないので関係ない

Calc_Potential_Flags.Non_Bonding_Intrachain（設定値：**1**）· · · 分子内の非結合間相互作用を計算するかどうかのフラグ。通常は 1 に設定

(e) 出力データ項目の設定
UDF Path 名 **Simulation_Conditions.Output_Flags** 以下のデータ値について説明する。

Statistics · · · 温度や圧力等の出力項目の選択。（選択された項目がデータおよびログファイルに出力される。UDF には選択の如何にかかわらずすべて出力される）

Structure.Position（設定値：**1**）
Structure.Velocity（設定値：**1**）
Structure.Force（設定値：**0**）· · · 各 Atom の座標、速度、力について出力の選択。座標および速度を出力する。

4. UDF の Save
各パラメータについて編集操作を施した場合、編集結果が反映されるように **GOURMET** の **File** → **Save** メニューにより UDF data を Save しなければならない。

5. **COGNAC** の起動

(**注意**)　エンジン起動の際には **Engine Manager** が起動していることを確認すこと。起動していない場合は、たとえば Windows の場合は、スタートメニュー **OCTA8.3** より **StartEngineManeger** を選択して起動しておくこと。

GOURMET Tool → **Engine Run...** メニューより開く **Engine Run panel** において、以下のパラメータを設定した後 **Run** ボタンにより **COGNAC** を起動する。以下に解説しないパラメータは、空欄あるいは、パネル立ち上げ時の状態のまでよい

- Engine: · · · 右端の [...] ボタンよりファイルブラウザを起動して、**COGNAC** 実行ファイル (“cognac92”) を選択する。
通常、実行ファイルは”$(PF_ENGINE)/bin/$(PF_ENGINEARCH)“ 以下にインストールされている。
（“$PF_ENGINE,$PF_ENGINEARCH” は各々、**OCTA** に含まれるシミュレーションプログラム群をインストールしたディレクトリ、シミュレーションプログラムを実行する環境 (linux,cygwin etc.) を指定する環境変数で、あらかじめ設定されている必要がある。詳細は「OCTA インストールマニュアル」参照）

- Working Dir: ··· ワーキングディレクトリとして適当なディレクトリ (“C:\temp” など) を指定。ただし、入力 UDF の存在するディレクトリを指定してはいけない
- Output UDF: ··· 出力 UDF として”A20B40A20_out.udf を” フルパスで指定。
- Summary UDF: ··· 計算経過を **Engine Control panel** でモニタするための UDF 名。任意の名前 (“summary.udf” など) を指定。これもフルパスで指定するのが望ましい。

なお、画面上部にある **Display Engine Control Window when Run Engine** のチェックをはずしておくと、エンジンが終了した後もログ出力のウインドウが閉じずに残るので、エラーメッセージ等の確認に有効

プログラムを実行すると **Engine Control panel** が立ち上がり、途中経過が表示される。初期構造緩和過程においてはデータが表示されないので、起動後データが表示されるまで若干時間がかかるので注意。

プログラムが終了すると、“A20B40A20_out.udf”（UDF 出力)、“A20B40A20_out.dat”（データ出力）が生成する。ただしデータ出力ファイルはワーキングディレクトリに生成し、プログラムが正常終了しても、転送されないので注意。

3.3 計算結果の表示と解析

ここでは、計算結果を **GOURMET** に読み込んで、分子構造の表示、解析を行う。

1. **Action** による分子構造の表示

   計算結果を **GOURMET** に読み込んで、**Action** 機能を用いて分子構造の表示を行う。

   (a) 出力 UDF の読み込み
   GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから
   “A20B40A20_out.udf” を選択する。その後分子構造を表示するために適当な Record に移動しておく

   (b) **Action** の起動
   画面のツリー部にある”A20B40A20_out.udf” アイコンを右クリックすると図 3.7 に示すように、登録されている Action コマンドが現れる。この、コマンドより **VIEWER_show...** を選択すると、図 3.8 に示すようなウインドウが立ち上がる。このウインドウにおいて、各パラメータを指定することにより様々な形式で分子構造の表示を行うことが出来る。

   パラメータの説明

   - **type** ··· 描画タイプ

     line/ball-stick/rod/volume より選択する。

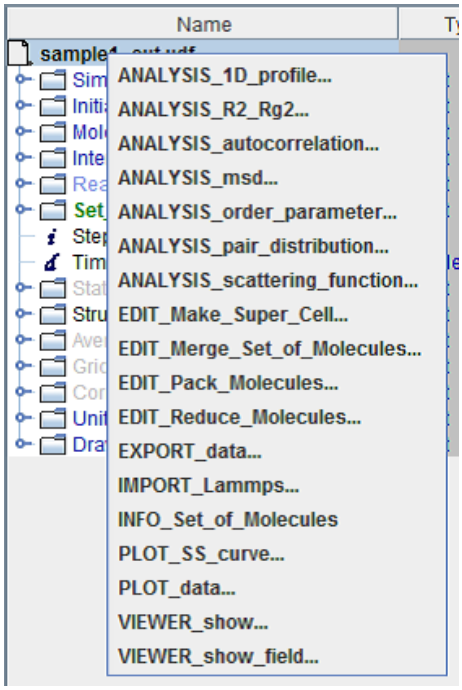


図 3.7: “A20B40A20_out.udf” の **Action** 選択時のイメージ

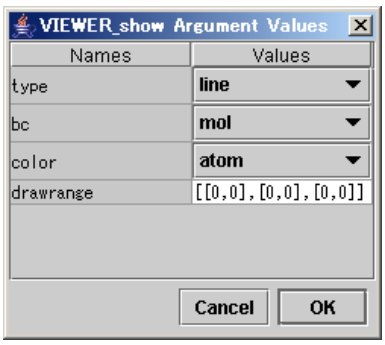


図 3.8: **VIEWER_show...** のパラメータ設定ウインドウ

- **bc** · · · 描画時の境界条件の設定。

  mol：分子の重心がユニットセル内に位置するように境界条件を作用させて表示する
  atom：各原子がすべてユニットセル内に位置するように境界条件を作用させて表示する
  off：境界条件を考慮せず、Atom の座標値で表示する

- **color** · · · 描画時の色指定

  atom：Atom type で色分けする
  bond：Bond type で色分けする
  mol：分子毎に色分けする。ただし OCTA がデフォルトで持つ色の設定には限りがあるので、多数の分子を表示するときは、同じ色が繰り返し使われる
  molname：Molecular name で色分けする

- **drawrange** ··· 描画時のユニットセル表示範囲
  基本のユニットセルに含まれる分子に加えて、周囲に存在するイメージも表示することが出来る。表示する範囲はリスト形式で与え、以下の順で範囲を指定する。
  [[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
  amin,amax,bmin... には各々a,b,c 軸方向に表示するセルの範囲を整数で指定する。例えば [[-1,1],[-1,1],[-1,1]] と指定した場合、a,b,c 軸すべて、基本セル内の分子と、-1 および 1 シフトしたイメージ分子が表示される。

図 3.9 に **Viewer** に表示される分子構造の例を示す。

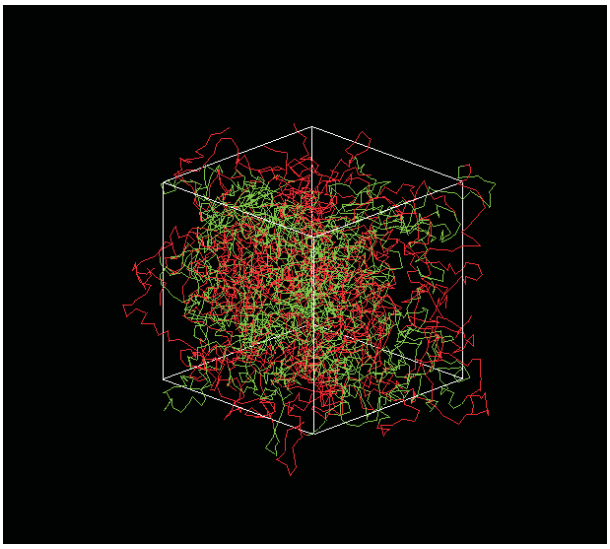


図 3.9: 分子構造表示の例

2. 慣性半径の解析

ここでは分子の慣性半径を計算する

(a) 出力 UDF の読み込み
GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから “A20B40A20_out.udf” を選択する。すでに読み込んでいれば必要ない。

(b) **Action** の起動
データ解析のため、画面下のスライドバーで任意の Record に移動しておく。
分子の描画と同様に、画面のツリー部にある”A20B40A20_out.udf” アイコンを右クリックして現れる、**Action** コマンドより、**ANALYSIS_R2_Rg2...** を選択すと、図 3.10 のようなウインドウが現れるので、**item** で **Rg** を選択して **OK** を押す。
すると、該当のレコードにおける構造を解析して、各分子の慣性半径 Rg^2 とその成分 Rg_x^2、Rg_y^2、Rg_z^2 を Python ログウインドウに出力する。

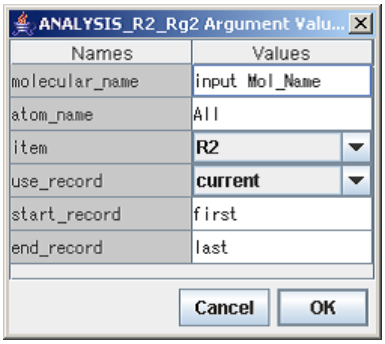


図 3.10: **ANALYSIS_R2_Rg2...** のパラメータ設定ウインドウ

3. 動径分布の解析

ここでは A-B atom 間の動径分布を計算する

(a) 出力 UDF の読み込み
GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから “A20B40A20_out.udf” を選択する。すでに読み込んでいれば必要ない。

(b) **Action** の起動
データ解析のため、画面下のスライドバーで任意の Record に移動しておく。
分子の描画と同様に、画面のツリー部にある”A20B40A20_out.udf” アイコンを右クリックして現れる、**Action** コマンドより、**ANALYSIS_pair_distribution...** を選択すと、図 3.11 のようなウインドウが現れるので、**OK** を押す。

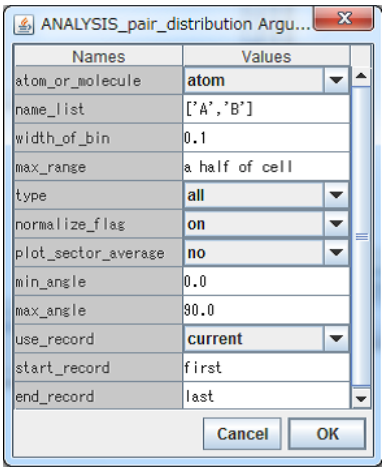


図 3.11: **ANALYSIS_pair_distribution...** のパラメータ設定ウインドウ

すると、該当のレコードにおける構造を解析して、A-B atom 間の動径分布を計算し、図 3.12 に示すような、プロットが表示される。
パラメータの詳細に関しては、§7.5 参照。

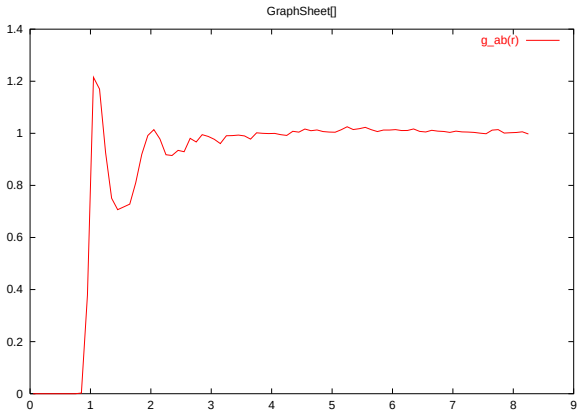


図 3.12: 動径分布のプロット例

第4章　適用事例

4.1　サンプルデーター一覧

サンプルデータおよびスクリプトは “sample” ディレクトリ以下に収録してある。

```
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
```

次に、用意されているサンプルデータに関して簡単に解説する。記されている UDF のパスは “sample” ディレクトリ以下のパスとしている。

4.1.1 Pentane

[Keyword] Pentane United Atom モデル / Nose Hoover NVT

- 計算モデル
  分子モデル ··· Unite Atom model [14]
  システムサイズ ··· pentane 50 分子
  初期構造 ··· リスタート
- 計算条件
  ソルバー ··· NVT_Nose_Hoover
  境界条件 ··· 3 次元周期境界条件
- サンプル UDF
  入力 UDF ··· “pentaneNVT/pentane50_in.udf”
  リスタート UDF ··· “pentaneNVT/pentane50_rst.udf”
  出力 UDF ··· “pentaneNVT/out/pentane50_out.udf”
  出力データファイル ··· “pentaneNVT/out/pentane50_out.dat”
  標準出力ファイル ··· “pentaneNVT/out/pentane50.log”
- 備考
  §4.2 参照

4.1.2 Poly(ethylene oxide) (PEO)

[Keyword] Electro static insteraction

Poly(ethylene oxide) と Li 塩からなる系におけるシミュレーション。PEO 上の各サイト（メチル、メチレン基はユナイテッドアトム近似）とイオン（Li^+，I^-） に電荷を与え、静電相互作用を考慮したシミュレーションを行っている。同時に、静電相互作用計算を、Ewald,PPPM,Reaction field,Cutoff 各々のアルゴリズム間で比較を行う。

PEO の O（負電荷）と Li イオン（正電荷）の静電相互作用によりクラウンエーテル状の配位をとる。

- 計算モデル
  分子モデル ··· United Atom Model
  システムサイズ ··· PEO：重合度 N = 12，5 分子。Li 塩：Li，I 各 5 原子。
  初期構造 ··· リスタート

- 計算条件
  ソルバー ··· NVT_Nose_Hoover
  境界条件 ··· 3 次元周期境界条件
  静電相互作用 ··· Ewald 法による計算 “peo_ewald”
  静電相互作用 ··· PPPM 法による計算 “peo_pppm”
  静電相互作用 ··· reaction field 法による計算 “peo_rf”
  静電相互作用 ··· cut off 法による計算 “peo_cutoff”

- サンプル UDF

  – Ewald 法
    入力 UDF ··· “peo/peo_ewald_in.udf”
    出力 UDF ··· “peo/out/peo_ewald_out.udf”
    出力データファイル ··· “peo/out/peo_ewald_out.dat”
    標準出力 ··· “peo/out/peo_ewald.log”

  – pppm 法
    入力 UDF ··· “peo/peo_pppm_in.udf”
    出力 UDF ··· “peo/out/peo_pppm_out.udf”
    出力データファイル ··· “peo/out/peo_pppm_out.dat”
    標準出力 ··· “peo/out/peo_pppm.log”

  – Reaction field 法
    入力 UDF ··· “peo/peo_rf_in.udf”
    出力 UDF ··· “peo/out/peo_rf_out.udf”
    出力データファイル ··· “peo/out/peo_rf_out.dat”
    標準出力 ··· “peo/out/peo_rf.log”

  – Cutoff 法
    入力 UDF ··· “peo/peo_cutoff_in.udf”
    出力 UDF ··· “peo/out/peoo_cutoff_out.udf”
    出力データファイル ··· “peo/out/peoo_cutoff_out.dat”
    標準出力 ··· “peo/out/peoo_cutoff.log”

  – 共通リスタート UDF ··· “peo/peo_rst.udf”

- 備考
  RATTLE による拘束。§4.3 参照

4.1.3 Liquid crystal(1): Gay-Berne potential

[Keyword] Gay-Berne potential / Two atoms site / Dipole reaction field

COGNAC においては、Interaction site を複数の原子から定義することが出来る。例えばここであげる Gay-Berne potential の様に、非球形のポテンシャルを設定するため、2 つの原子から Interaction Site を定義して、それらの原子の座標から、ポテンシャルの中心、ベクトル等を計算して、実際のポテンシャルを求めることが出来る。

- 計算モデル
  分子モデル ··· Gay-Berne potential

システムサイズ · · · 2 原子分子（2 原子で 1 つの Gay-Berne interaction site を定義する） 256 分子
初期構造 · · · リスタート

- 計算条件
  ソルバー　· · · NVE
  境界条件　· · · 3 次元周期境界条件
  静電相互作用　· · · Reaction field 法による Dipole-Dipole 静電相互作用を計算する

- サンプル UDF
  入力 UDF · · · “GBpotential/GBpotential_in.udf”
  出力 UDF · · · “GBpotential/out/GBpotential_out.udf”
  出力データファイル · · · “GBpotential/out/GBbpotential_out.dat”
  標準出力 · · · “GBpotential/out/GBbpotential.log”

- 備考
  RATTLE により bond length を拘束している。そのため Bond potential は計算していない

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

Gay–Berne ポテンシャルと Lennard-Jones ポテンシャルを組み合わせたモデルのテスト

- 計算モデル
  システムサイズ · · · 5CB(4-n-pentyl-4’-cyanobiphenyl) 256 分子
  (4-methyl-4’-cyanobiphenyl 部分は楕円に近似し、アルキルテールの部分は球の連鎖で表現する)
  ポテンシャル · · · angle potential、torsion potential、Gay Berne potential、Lennard-Jones potential
  (サンプルにある、ポテンシャルパラメータはおおよその見積もりで、十分に正確ではない [46, 47])
  初期構造 · · · FCC 型のパッキング構造を作成

- 計算条件
  ソルバー · · · NVT_Berendsen
  境界条件 · · · 3 次元周期境界条件

- サンプル UDF
  入力 UDF · · · “GBLJpotential/GBLJpotential_in.udf”
  出力 UDF · · · “GBLJpotential/out/GBLJpotential_out.udf”
  出力データファイル · · · “GBLJpotential/out/GBLJpotential_out.dat”
  標準出力 · · · “GBLJpotential/out/GBLJpotential.log”

- 備考
  RATTLE により bond length を拘束している。そのため Bond potential は計算していない

4.1.5 Thermoplastic elastomer

[Keyword] Altanative block copolymer / Uniaxial elongation / Shear flow

交互ブロック共重合体である熱可塑性エラストマーのモデルに、伸長、あるいは、ずり流動を与え、応力変化を計算する例。

ポテンシャルはハードセグメント (A) 間にのみ引力項を与えて、その他は斥力項のみを与えることによりミクロ相分離を誘起している。ただしサンプルデータにおいては、初期構造において十分ミクロ相分離構造を取っていないのでゴム弾性の発現は弱い。

- 計算モデル
  分子モデル ··· Bead-Spring (harmonic bond) model
  システムサイズ ··· (A3B21)6 ブロックコポリマー 48 分子
  初期構造 ··· リスタート
- 計算条件
  ソルバー ··· NVE
  境界条件 ··· 3 次元周期境界条件
  変形
  - ユニットセル変形による 1 軸伸張
  - SLLOD+Lees-Edwards 境界条件によるずり流動
- サンプル UDF
  - 伸張変形
    入力 UDF ··· “tpe/tpe_in.udf”
    出力 UDF ··· “tpe/out/tpe_out.udf”
    出力データファイル ··· “tpe/out/tpe_out.dat”
    標準出力 ··· “tpe/out/tpe.log”
  - ずり流動
    入力 UDF ··· “tpe/tpe_shear_in.udf”
    出力 UDF ··· “tpe/out/tpe_shear_out.udf”
    出力データファイル ··· “tpe/out/tpe_shear_out.dat”
    標準出力 ··· “tpe/out/tpe_shear.log”
  - 共通リスタート UDF ··· “tpe/tpe_rst.udf”

4.1.6 Thermo plastic elastomer (2)

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

BCC 球状ミクロ相分離構造を示す、ABA トリブロックコポリマーに伸張を与えて応力変化を計算する例。SUSHI により求められたミクロ相分離構造より、Density biased Monte Carlo 法により初期構造を発生している。計算の詳細は、文献 [48] 参照

- 計算モデル
  分子モデル ··· Bead-Spring model
  システムサイズ ··· A5B73A5 トリブロックコポリマー 347 分子
  初期構造 ··· リスタート
- 計算条件
  ソルバー ··· NVT Kremer Grest
  境界条件 ··· 3 次元周期境界条件
  伸張変形 ··· ユニットセル変形による 1 軸伸張

- サンプル UDF 入力 UDF ··· “tpe2/a5b73a5_in.udf”
  リスタート UDF ··· “tpe2/a5b73a5_rst.udf” 出力 UDF ··· “tpe2/out/a5b73a5_out.udf”
  出力データファイル ··· “tpe2/out/a5b73a5_out.dat”
  標準出力 ··· “tpe2/out/a5b73a5.log”

4.1.7 Solid wall

[Keyword] bead-spring モデル / 2D 周期境界条件 / LJ type flat wall potential / LJ atomic wall potential

上下を固体壁にはさまれた、溶融高分子のシミュレーション例。壁のポテンシャルとして Lennard-Jones potential タイプの解析的に与えた平滑なポテンシャルと、壁上に Lennard-Jones potential の中心を実際に配置した物とを比較している。適用例としては文献 [49] 参照

- 計算モデル
  分子モデル ··· Kremer Grest type [25] bead-spring model
  システムサイズ ··· 重合度 N = 40、100 分子
  ユニットセル ··· $11.71 \times 11.71 \times 35.0$
  初期構造 ··· リスタート

- 計算条件
  ソルバー ··· NVT_Kremer_Grest
  境界条件 ··· xy 方向に 2 次元周期境界条件
  外場
  - Flat wall *cdots* xy 平面上に LJ type flat wall potential [31]
  - Atomistic wall *cdots* xy 平面上に LJ atomistic wall potential [31]

- サンプル UDF
  - Flat wall
    入力 UDF ··· “wall/wall35_in.udf”
    出力 UDF ··· “wall/out/wall35_out.udf”
    出力データファイル ··· “wall/out/wall35_out.dat”
    標準出力 ··· “wall/out/wall35.log”
  - Atomistic wall
    入力 UDF ··· “wall/wall35_atom_in.udf”
    出力 UDF ··· “wall/out/wall35_atom_out.udf”
    出力データファイル ··· “wall/out/wall35_atom_out.dat”
    標準出力 ··· “wall/out/wall35_atom.log”

  共通リスタート UDF ··· “wall/wall35_rst.udf”

- 備考
  Thermal noise による温度制御を行っているため、出力結果は完全に同一にはならないので注意

4.1.8 Graft chain

[Keyword] End grafted chain / bead-spring モデル / 2D 周期境界条件 / LJ type flat wall potential

- 計算モデル
  分子モデル ··· Kremer Grest type [25] bead-spring model
  システムサイズ ··· 重合度 N = 40、100 分子
  ユニットセル ··· 20.0 × 20.0 × 50.0
  初期構造 ··· ランダム（末端位置指定）

- 計算条件
  ソルバー ··· NVT_Kremer_Grest
  境界条件 ··· xy 方向に 2 次元周期境界条件
  外場 ··· z 方向に LJ type Wall potential [31]
  Constraint atom ··· z=1.0 の xy 平面上に末端原子を固定

- サンプル UDF
  入力 UDF ··· “graft/graft_in.udf”
  出力 UDF ··· “graft/out/graft_out.udf”
  出力データファイル ··· “graft/out/graft_out.dat”
  標準出力 ··· “graft/out/graft.log”

- 備考
  初期構造をランダムに発生させる際、**Initial_Structure.Generate_Method.Random.Fix_End_Position**[]
  で末端の座標を指定することができる。この機能を用いてグラフト鎖の末端の座標を z=1 平面上の正方
  格子点に配置する。

4.1.9 Periodic chain

[Keyword] Polyethylene United Atom モデル / Periodic Chain

- 計算モデル
  分子モデル ··· Unite Atom model [14]
  システムサイズ ··· C20 からなる無限長鎖 20 分子

- 計算条件
  ソルバー ··· NVE → NPT Parrinello Rahman アルゴリズム
  境界条件 ··· 3 次元周期境界条件。Periodic chain を考慮

- サンプル UDF

  1. 初期構造生成 & NVE Dynamics
     入力 UDF ··· “infiniteChain/inf/infiniteChain_in.udf”
     分子構造データ UDF ··· “infiniteChain/infiniteChain_rst.udf”
     出力 UDF ··· “infiniteChain/out/infiniteChain_out.udf”
     出力データファイル ··· “infiniteChain/out/infiniteChain_out.dat”
     標準出力 ··· “infiniteChain/out/infiniteChain.log”
  2. NPT Parrinello Rahman Dynamics
     入力 UDF ··· “infiniteChain/inf_PR/infiniteChainPR_in.udf”

リスタート UDF ··· “infiniteChain/out/infiniteChain_out.udf” (前ステップで作成したもの)
出力 UDF ··· “infiniteChain/out/infiniteChainPR_out.udf”
出力データファイル ··· “infiniteChain/out/infiniteChainPR_out.dat”
標準出力 ··· “infiniteChain/out/infiniteChainPR.log”

- 備考
  ステップ 2 ではステップ 1 の最終結果を用いて、NPT アンサンブルでシミュレーションを継続している。

4.1.10 Table potential

ポテンシャルを関数の代りにテーブルデータで与える場合のテスト

- 計算モデル
  分子モデル ··· Unite Atom model [14](各ポテンシャルをテーブル化)
  システムサイズ ··· pentane 50 分子
  初期構造 ··· リスタート

- 計算条件
  ソルバー ··· NVT_Nose_Hoover
  境界条件 ··· 3 次元周期境界条件

- サンプル UDF
  入力 UDF ··· “tablepotential/pentane50_table_in.udf”
  リスタート UDF ··· “tablepotential/pentane50_table_rst.udf”
  出力 UDF ··· “tablepotential/out/pentane50_table_out.udf”
  出力データファイル ··· “tablepotential/out/pentane50_table_out.dat”
  標準出力 ··· “tablepotential/out/pentane50_table.log”
  Bond Potential Table UDF ··· “tablepotential/bond_table.udf”
  Angle Potential Table UDF ··· “tablepotential/angle_table.udf”
  Torsion Potential Table UDF ··· “tablepotential/torsion_table.udf”
  Non-Bonding Potential Table UDF ··· “tablepotential/nb_table.udf”

- 備考
  §4.5 参照

4.1.11 Crystal generator

結晶構造データよりの初期構造の作成

- 計算モデル
  分子モデル ··· Unite Atom model [14]
  システムサイズ ··· icosane 8 分子
  初期構造 ··· 結晶構造ビルダーにより発生 [50]

- 計算条件
  ソルバー ··· NVT_Nose_Hoover
  境界条件 ··· 3 次元周期境界条件

- サンプル UDF
  入力 UDF ··· “crystal/pemodel20_crystal_in.udf”
  出力 UDF ··· “crystal/out/pemodel20_crystal_out.udf”
  出力データファイル ··· “crystal/out/pemodel20_crystal_out.dat”
  標準出力 ··· “crystal/out/pemodel20_crystal.log”
  結晶構造データ UDF ··· “crystal/cryst_xtl.udf”

- 備考

4.1.12 Comparison of NPT algorithm

[Keyword] NPT アルゴリズムの比較 / United Atom モデル / Pentane

- 計算モデル
  分子モデル ··· Unite Atom model [14]
  システムサイズ ··· pentane 50 分子
  初期構造 ··· リスタート

- 計算条件
  境界条件　··· 3 次元周期境界条件
  温度制御 ··· 拡張ハミルトニアン法においては Nose-Hoover, ルーズカップリング法においては温度制御もルーズカップリング法により行っている
  Cutoff 距離　··· Lennard-Jones potential の cut off は 2.5σ、tail correction により、エネルギー、ビリアルの補正を行っている

- サンプル UDF

  – 共用 UDF
    リスタート UDF ··· “NPT/rst_files/npt_rst.udf”

  – Andersen アルゴリズム（拡張ハミルトニアン法、等方セル）
    入力 UDF ··· “NPT/andersen/andersen_in.udf”
    出力 UDF ··· “NPT/out/andersen_out.udf”
    出力データファイル ··· “NPT/out/andersen_out.dat”
    標準出力 ··· “NPT/out/andersen.log”

  – Parrinello-Rahman アルゴリズム（拡張ハミルトニアン法、非等方セル）
    入力 UDF ··· “NPT/prnh/prnh_in.udf”
    出力 UDF ··· “NPT/out/prnh_out.udf”
    出力データファイル ··· “NPT/out/prnh_out.dat”
    標準出力 ··· “NPT/out/prnh.log”

  – Berendsen アルゴリズム（ルーズカップリング法、等方セル）
    入力 UDF ··· “NPT/berendsen/berendsen_in.udf”
    出力 UDF ··· “NPT/out/berendsen_out.udf”
    出力データファイル ··· “NPT/out/berendsen_out.dat”
    標準出力 ··· “NPT/out/berendsen.log”

  – Brown-Clarke アルゴリズム（ルーズカップリング法、非等方セル）
    入力 UDF ··· “NPT/BC/BC_in.udf”

出力 UDF ··· “NPT/out/BC_out.udf”
出力データファイル ··· “NPT/out/BC_out.dat”
標準出力 ··· “NPT/out/BC.log”

- 備考
温度制御、圧力制御のためのパラメータは、システムサイズ、材料物性等により最適値が異なり、ここで用いているサンプルの値も最適値である保証は無いが、一応パラメータ選択の目安をあげておく。

  - Nose-Hoover **Q** param
  経験的に 20 に設定。文献 [51] と比較して単位変換すると $Q = 40(kJ/mol)(ps)^2$ となりやや大きい
  - **Cell Mass**
  経験的にシステムのトータル質量と同じ (pentane 50 分子, $M_{total} = 250$) に設定。しかし、Parrinello-Rahman アルゴリズムの場合、セル角度の変形が非常に大きくなるので、もっと大きな値の方がよいかも知れない
  - **tauT**
  文献 [16] より $\tau_T = 0.2$(ps) に設定
  - Berendsen **tauP**
  文献 [16] より $\beta = 4.9 \times 10^{-5}\text{bar}^{-1}$ とおき設定。$\tau_P/\beta = 6.6$ より $\tau_P = 0.1$(ps)
  - Brown-Clarke **tauP**
  文献 [20] より $\tau_P = 5.28 \times 10^6(\text{Pasm}^{-1})$ に設定

4.1.13 Test of tail correction

Lennard-Jones potential の cut off 距離を変えて、Tail Correction のテストを行っている

- 計算モデル
分子モデル ··· Unite Atom model [14]
システムサイズ ··· pentane 50 分子
初期構造 ··· リスタート

- 計算条件
境界条件 ··· 3 次元周期境界条件
ソルバー ··· Andersen,Nose-Hoover による温度、圧力制御を行っている。
Cutoff 距離 ··· Lennard-Jones potential の cut off は 1.5σ、tail correction により、エネルギー、ビリアルの補正を行っている

- サンプル UDF
入力 UDF ··· “tailCorrection/andersen_short_in.udf”
リスタート UDF ··· “tailCorrection/npt_rst.udf” （Comparison of NPT algorithm と同じ初期データ）
出力 UDF ··· “tailCorrection/out/andersen_short_out.udf”
出力データファイル ··· “tailCorrection/out/andersen_short_out.dat”
標準出力 ··· “tailCorrection/out/andersen_short.log”

- 備考
Comparison of NPT algorithm で行っている、andersen($cutoff = 2.5\sigma$) と結果を比較すると、Tail Correction が機能しているため、non bonding energy、密度がほぼ同じ値を与えることがわかる。

4.1.14 Test of RATTLE

RATTLE algorithm を用いる際の time step のテストを行っている

- 計算モデル
  分子モデル ··· Unite Atom model [14]
  システムサイズ ··· pentane 50 分子
  初期構造 ··· リスタート
- 計算条件
  境界条件 ··· 3 次元周期境界条件
  ソルバー ··· NVE, 温度スケール無し
- サンプル UDF
  - 共用 UDF
    リスタート UDF ··· “RATTLE/rattle_rst.udf”
  - RATTLE on
    入力 UDF ··· “RATTLE/rattle_*dt*_in.udf”
    出力 UDF ··· “RATTLE/out/rattle_*dt*_out.udf”
    出力データファイル ··· “RATTLE/out/rattle_*dt*_out.dat”
    標準出力 ··· “RATTLE/out/rattle_*dt*.log”
  - RATTLE off
    入力 UDF ··· “RATTLE/norattle_*dt*_in.udf”
    出力 UDF ··· “RATTLE/out/norattle_*dt*_out.udf”
    出力データファイル ··· “RATTLE/out/norattle_*dt*_out.dat
    標準出力 ··· “RATTLE/out/norattle_*dt*.log”

  dt はタイムステップを示す。$\Delta t = dt \times 10^{-3} = dt \times 2.0$(fs)
- 備考
  NVE の計算結果より、ハミルトニアンの保存のタイムステップ依存性がわかる。
  RATTLE off の初期構造にも RATTLE on と同じリスタートデータを用いているので、bond potential が若干非平衡状態に有ることに注意

4.1.15 Minimization

[Keyword] Pentane United Atom モデル / Minimize

- 計算モデル
  分子モデル ··· Unite Atom model [14]
  システムサイズ ··· pentane 50 分子
  初期構造 ··· Random
- 計算条件
  ソルバー ··· Minimize, Cascade option
  境界条件 ··· 3 次元周期境界条件

- サンプル UDF
  入力 UDF ··· “Minimize/minimize_in.udf”
  出力 UDF ··· “Minimize/out/minimize_out.udf”
  出力データファイル ··· “Minimize/out/minimize_out.dat”
  標準出力 ··· “Minimize/out/minimize.log”

- 備考
  ランダムな初期構造を発生させて、Steepest Descent - Conjugate Gradient の組み合わせでエネルギー極小化を行う。
  COGNAC の Conjugate Gradient アルゴリズムの実装はさほどインテリジェントではないので、Cascade option では、極小点付近における収束性はあまりよくない。(MD で発散しない初期構造作成程度には十分使える)

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

[Keyword] Density biased Monte Carlo / Density biased potential

SUSHI によるセグメント密度よりブロックコポリマーのラメラ構造を生成する

- 計算モデル
  分子モデル ··· Kremer Grest type [25] bead-spring model
  システムサイズ ··· A20B40A20 トリブロックコポリマー、50 分子
  ユニットセル ··· $10.98 \times 10.98 \times 39$
  初期構造 ··· Density biased Monte Carlo

- 計算条件
  ソルバー ··· NVT_Kremer_Grest
  境界条件 ··· 3 次元周期境界条件
  外場 ··· Density biased potential

- サンプル UDF
  入力 UDF ··· “block/A20B40A20_in.udf”
  分子構造データ UDF ··· “block/A20B40A20_in_str.udf”
  出力 UDF ··· “block/out/A20B40A20_out.udf”
  出力データファイル ··· “block/out/A20B40A20_out.dat”
  標準出力 ··· “block/out/A20B40A20.log”
  SUSHI 出力ファイル ··· “block/A20B40A20_sushi_out.udf”

- 備考 §2.7.1 §2.6.5 §4.4 参照

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

[Keyword] Density biased Monte Carlo / Staggered reflective boundary condition

SUSHI によるセグメント密度よりポリマーブレンドの界面構造を生成する。

- 計算モデル
  分子モデル ··· Kremer Grest type [25] bead-spring model

システムサイズ ··· A100/B100 homopolymer blend。分子数 $M_A = M_B = 40$
ユニットセル ··· $14.00 \times 14.00 \times 48.00$
初期構造 ··· Density biased Monte Carlo

- 計算条件
  ソルバー ··· NVT_Kremer_Grest
  境界条件 ··· スタガード反射境界条件 (Z 軸方向)

- サンプル UDF
  入力 UDF ··· “blend/blend_de01_in.udf”
  分子構造データ UDF ··· “blend/blend100_silk_str.udf”
  出力 UDF ··· “blend/out/blend_de01_out.udf”
  出力データファイル ··· “blend/out/blend_de01_out.dat”
  標準出力 ··· “blend/out/blend_de01.log”
  SUSHI 出力ファイル ··· “blend/sushi_chiN200_out.udf”

- 備考 §2.7.1 §2.6.5 §4.4 参照

4.1.18 Depletion: Zoom in from SUSHI (3)

[Keyword] Density biased Monte Carlo / Staggered reflective boundary condition / Solid wall

高分子／低分子のブレンドの固体壁近傍における濃度分布の偏りを SUSHI によるセグメント密度より生成する。

- 計算モデル
  分子モデル ··· Kremer Grest type [25] bead-spring model
  システムサイズ ··· 重合度 $N_A = 100$、分子数 $M_A = 10$ と $N_B = 1$、$M_B = 100$ のブレンド
  ユニットセル ··· $6.36 \times 6.36 \times 33.0$
  初期構造 ··· Node density Monte Carlo により発生

- 計算条件
  ソルバー ··· NVT_Kremer_Grest
  境界条件 ··· xy 方向に 2 次元周期境界条件。z 方向はバルク側 ($z = Z_{max}$) の境界条件は Staggered reflective boundary condition
  外場 ··· z=0 の面に LJ type Wall potential [31]

- サンプル UDF
  入力 UDF ··· “depletion/depletion_in.udf”
  出力 UDF ··· “depletion/out/depletion_out.udf”
  出力データファイル ··· “depletion/out/ depletion_out.dat”
  標準出力 ··· “depletion/out/depletion.log”
  SUSHI 出力 UDF ··· “depletion/depletion_sushi_out.udf”

- 備考
  §2.7.1 §2.8 参照

4.1.19 Semi-crystalline lamella

結晶ラメラ初期構造を作成する

- 計算モデル

  1. ビーズースプリングモデル
     分子モデル ··· Kremer Grest type [25] bead-spring model
     システムサイズ ··· 重合度 n = 1200, 4 分子
     ユニットセル ··· $15.89 \times 15.89 \times 20$
  2. ユナイティッドアトムモデル
     分子モデル ··· 直鎖アルカンユナイティッドアトム [52]
     システムサイズ ··· 重合度 (CH_2 unit)n = 1200、4 分子
     ユニットセル ··· $10.89 \times 10.89 \times 15$

  初期構造 ··· Lamella generator

- 計算条件ソルバー ··· NVE
  境界条件 ··· 3 次元周期境界条件

- サンプル UDF

  1. ビーズースプリングモデル入力 UDF ··· “lamella/bs_lamella_in.udf”
     分子構造データ UDF ··· “lamella/bs_lamella_in_str.udf”
     出力 UDF ··· “lamella/out/bs_lamella_out.udf”
     出力データファイル ··· “lamella/out/bs_lamella_out.dat”
     標準出力 ··· “lamella/out/bs_lamella.log”
  2. ユナイティッドアトムモデル入力 UDF ··· “lamella/ua_lamella_in.udf”
     分子構造データ UDF ··· “lamella/ua_lamella_in_str.udf”
     出力 UDF ··· “lamella/out/ua_lamella_out.udf”
     出力データファイル ··· “lamella/out/ua_lamella_out.dat”
     標準出力 ··· “lamella/out/ua_lamella.log”

- 備考 §2.7.1 §4.7 参照

4.1.20 Reaction

化学反応モデルによる結合の生成をシミュレーションする

- 計算モデル分子モデル ··· Harmonic bond+LJ non-bonding interaction type ビーズースプリングモデル
  初期分子アーキテクチャー ··· 単原子分子 A/B ブレンド
  分子数 $M_A = M_B = 50$
  初期構造 ··· Random
  単原子分子 A/B ブレンドを初期構造とした、A-B.A-A,B-B 間の結合の生成、解離のシミュレーション。

- 計算条件ソルバー ··· Kremer_Grest type thermostat
  境界条件 ··· 3 次元周期境界条件

- サンプル UDF
  入力 UDF ··· “reaction/react_in.udf”
  分子構造データ UDF ··· “reaction/react_in_str.udf”
  出力 UDF ··· “reaction/out/react_out.udf”
  出力データファイル ··· “reaction/out/react_out.dat”
  標準出力 ··· “reaction/out/react.log”

- 備考 §2.9 §4.6 参照

4.1.21 Polymerization

化学反応モデルによる重合反応をシミュレーションする

- 計算モデル分子モデル ··· Harmonic bond+LJ non-bonding interaction type ビーズ－スプリングモデル
  初期分子アーキテクチャー ··· モノマー M、開始剤 I を含む
  分子数 $\mathrm{M_M} = 1000, \mathrm{M_I} = 5$
  初期構造 ··· Random
  開始剤とモノマーとの反応から始まる重合反応のシミュレーション。

- 計算条件ソルバー ··· Kremer_Grest type thermostat
  境界条件 ··· 3 次元周期境界条件

- サンプル UDF
  入力 UDF ··· “polymerization_in.udf”
  出力 UDF ··· “polymerization/out/polymerization_out.udf”
  出力データファイル ··· “polymerization/out/polymerization_out.dat”
  標準出力 ··· “polymerization/out/polymerization.log”

- 備考 §2.9 参照

4.1.22 DPD

[keyword] DPD / diblock copolymer / microphase separation

Dissipative particle dynamics により、AB ジブロックコポリマーのミクロ相分離シミュレーションを行う。モデル、パラメータの詳細は文献 [13] 参照

- 計算モデル
  分子モデル ···　A5B5 ジブロックコポリマー
  システムサイズ ··· A5B5 ジブロックコポリマー、分子数 2,400
  ユニットセル ··· $20.0 \times 20.0 \times 20.0$
  初期構造 ··· ランダム

- 計算条件
  ソルバー ··· DPD
  境界条件 ··· 3 次元周期境界条件

- サンプル UDF
  入力 UDF ··· “DPD/A5B5_in.udf”
  出力 UDF ··· “DPD/out/A5B5_out.udf”
  出力データファイル ··· “DPD/out/ A5B5_out.dat”
  標準出力 ··· “DPD/out/A5B5.log”

- 備考
  §2.4 参照

4.1.23 External flow

[Keyword] Velocity field / SLLOD

Muffin_phaseseparation により計算された流れ場を、SLLOD アルゴリズムにより Atom に作用させる。このサンプルでは「Muffin ユーザーズマニュアル」§5.1.8 の例題に類似した、壁のずりにより生じるずり流動を読み込む。

- 計算モデル
  分子モデル ··· Kremer Grest type [25] bead-spring model
  システムサイズ ··· 重合度 $N = 10$、分子数 $M = 650$
  ユニットセル ··· $16.0 \times 16.0 \times 32.0$
  初期構造 ··· ランダム

- 計算条件
  ソルバー ··· NVE
  境界条件 ··· xy-2 次元周期境界条件
  外場
  - xy 平面に LJ wall
  - 速度場

- サンプル UDF
  入力 UDF ··· “externalflow/externalflow_in.udf”
  Muffin 出力 UDF ··· “Muffin_shear_out.udf”
  出力 UDF ··· “externalflow/out/externalflow_out.udf”
  出力データファイル ··· “externalflow/out/ externalflow_out.dat”
  標準出力 ··· “externalflow/out/externalflow.log”

- 備考
  §2.6.5 参照

4.1.24 Stress autocorrelation

[Keyword] On the fly

On the fly stress autocorrelation の解析を行うサンプル。

- 分子モデル ··· Kremer Grest type [25] bead-spring model
  システムサイズ ··· 重合度 $N = 40$、分子数 $M = 100$
  初期構造 ··· ランダム

- 計算条件
  ソルバー ··· NVT_Kremer_Grest
  境界条件 ··· 3 次元周期境界条件

- サンプル UDF
  入力 UDF ··· “correlation/n40_100_in.udf”
  リスタート UDF ··· “correlation/n40_100_rst.udf”
  出力 UDF ··· “correlation/out/n40_100_out.udf”
  出力データファイル ··· “correlation/out/n40_100_out.dat
  標準出力 ··· “correlation/out/n40_100.log”

- 備考
  §2.11 参照

4.1.25 MDSCF

[Keyword] MDSCF

Milano and Kawakatsu らの MDSCF 法 [53] によるブロックコポリマーのミクロ相分離構造形成のテスト。温度制御等詳細な方法はオリジナルのアルゴリズムと異なる部分があるので注意。

- 分子モデル ··· Kremer Grest type [25] bead-spring model
  システムサイズ ··· A15B15 ジブロックコポリマー、分子数 320
  ユニットセル ··· $22.436 \times 22.436 \times 22.436$
  初期構造 ··· ランダム

- 計算条件
  ソルバー ··· NVT_Kremer_Grest
  境界条件 ··· 3 次元周期境界条件
  外場

  – Density_Filed ··· Intearction Site A に対して Site B の濃度に従ったポテンシャルを与える。
  – Density_Filed ··· Intearction Site B に対して Site A の濃度に従ったポテンシャルを与える。
  – Total_Density_Constrain

  （補足）Density_Field で UDF_Name を指定しない場合、Simulation_Conditions.Density_Output で作成する Field が用いられる。

- サンプル UDF
  入力 UDF ··· “MDSCF/a15b15_mdscf_in.udf”
  出力 UDF ··· “MDSCF/out/a15b15_mdscf_out.udf”
  出力データファイル ··· “MDSCF/out/a15b15_mdscf_out.dat”
  標準出力 ··· “MDSCF/out/a15b15_mdscf.log”

4.2 （事例 I）ユナイティッドアトムモデルによる n-アルカンのシミュレーション

ここでは、ユナイティッドアトムモデルによる n-アルカンの溶融状態のシミュレーションの例を示す。
使用するファイルは特に指定するものを除いて、“sample/pentaneNVT” 以下にある。

4.2.1 SILK による入力データ作成

SILK は Python により記述された、**COGNAC** の入力ファイルを作成する補助ツールである。詳細は第 6 章を参照のこと。

SILK を実行する為には、シミュレーションの条件ならびにポテンシャルが記述された **SILK** 用入力 UDF が必要である。**SILK** 用入力 UDF は、**COGNAC** 入力 UDF とほぼ同じ構造を持つ。雛型として用意されている “potential_map.udf” をとって説明する。

1. **SILK** 入力ファイル（ “potential_map.udf” ）

   SILK 入力ファイル（ “python/silk/sample/potential_map.udf” ）を参照する。“potential_map.udf” は複数のレコード（UDF の基本データ単位）からなっている。それぞれのレコード毎にシミュレーションを行うために必要な条件、ならびにポテンシャルのパラメータのサンプルが格納されている。ここでは、最初のレコード（Record Number は **0**、Record Label は “UA_n_ALKANE”）を使用する。

2. UDF（ “potential_map.udf”）の Open

   GOURMET を起動し、UDF（ “potential_map.udf” ）を Open する。Records モードで表示させると、画面には Record Number が **0** のデータが表示されるので、右下の窓に Record Label である “UA_n_ALKANE” が表示されていることを確認する。

3. 分子の作成（**SILK** スクリプトの Load と編集ならびに実行）

   ここでは、Pentane 50 分子からなるシステムを作成する手順について説明する。
   なお、このシステムは **Action SILK** コマンド **SILK_CREATE_LinearPolymer_Semiatomic** を用いても作成することが出来る。コマンドの詳細は第 6 章参照。

   (a) **SILK** スクリプトの Load
   GOURMET 上で “python/silk/sample/silk_use_pentane.py” を Load する。Python 窓に Load されたスクリプトが表示されていることを確認し、スクリプトの上部を参照する。

   (b) Load された **SILK** スクリプトの編集
   Load されたスクリプトの上部にユーザーが編集すべきセクション（**SECTION “USER DEFINITION”**）が記述されている。

   - 出力先
     “USER DEFINITION” 内の **SUBSECTION “outputpath”** について説明する。
     *パイソン文法では、文頭に#を記述すると、その行はコメントととなる。

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

self.engine.outDir には出力先のディレクトリを指定する。

self.engine.cognacFileName には出力ファイル名（UDF 拡張子：.udf を除く）を指す。この例では “pentane50_in.udf” という **SILK** の出力、すなわち **COGNAC** 入力ファイルが作成される。

self.engine.prjName には出力ファイルのプロジェクト名を指す。

self.engine.fileNumCom には出力ファイルを分割する（トポロジーを記述したファイルとその他の条件を記述したファイルに分割する）か否かのコマンドを記述する。
self.engine.fileNumCom=”TWO_FILES” とした場合、分割された二つのファイルにまとめられる。

- 分子の定義
  “USER DEFINITION” 内の **SUBSECTION “system”** について説明する。
  記述したスクリプトは以下の通り。

```
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)
```

最下行で Call されている関数 **makeLinPolym(. . .)** は直鎖状分子を作成する関数である。
self.engine. は、この関数を使用する為に不可欠な記述である。
この関数 **makeLinPolym(. . .)** が参照している引数について、次に説明する。

name は分子に設定される名前である。

numAtom は 1 分子当りの Atom の個数である。

numMol は分子の本数である。

atomName は pentane 分子内の各 Atom につけられる名前である。

atomTypeName は、Atom のタイプの属性を指す。
UDF Path 名 **Molecular_Attributes.Atom_Type**[] の配列化されたデータの内、データ **Name** に合致するものが参照される（ここでは UDF Path 名 **Molecular_Attributes.Atom_Type**[**0**] のデータが参照される）。

図 4.1 に入力 UDF とスクリプトの関連を示す。

- UDF の Save
  各パラメータについて編集操作を施した場合、編集結果が反映されるように **GOURMET** の **File** → **Save** メニューにより UDF data を Save しなければならない。

bondTypeName は、Bond（結合長に基づくポテンシャル）のタイプの属性を指す。
UDF Path 名 **Molecular_Attributes.Bond_Potential**[] の配列化されたデータの内、データ **Name** に合致するものが参照される（ここでは UDF Path 名 **Molecular_Attributes.Bond_Potential**[**0**] のデータが参照される）。

図 4.2 に入力 UDF とスクリプトの関連を示す。

angleTypeName は、Angle（結合角に基づくポテンシャル）のタイプの属性を指す。
UDF Path 名 **Molecular_Attributes.Angle_Potential**[] の配列化されたデータの内、データ **Name** に合致するものが参照される（ここでは UDF Path 名 **Molecular_Attributes.Angle_Potential**[**0**] のデータが参照される）。

torsionTypeName は、Torsion（二面角に基づくポテンシャル）のタイプの属性を指す。
UDF Path 名 **Molecular_Attributes.Torsion_potenPotential**[] の配列化されたデータの内、データ **Name** に合致するものが参照される
（ここでは UDF Path 名 **Molecular_Attributes.Torsion_Potential**[**0**] のデータが参照される）。

interactionSiteTypeName は、InteractionSite （分子間相互作用に関するサイト）のタイプの属性を指す。
UDF Path 名 **Molecular_Attributes.Interaction_Site_Type**[] のデータ名 **Name** が参照

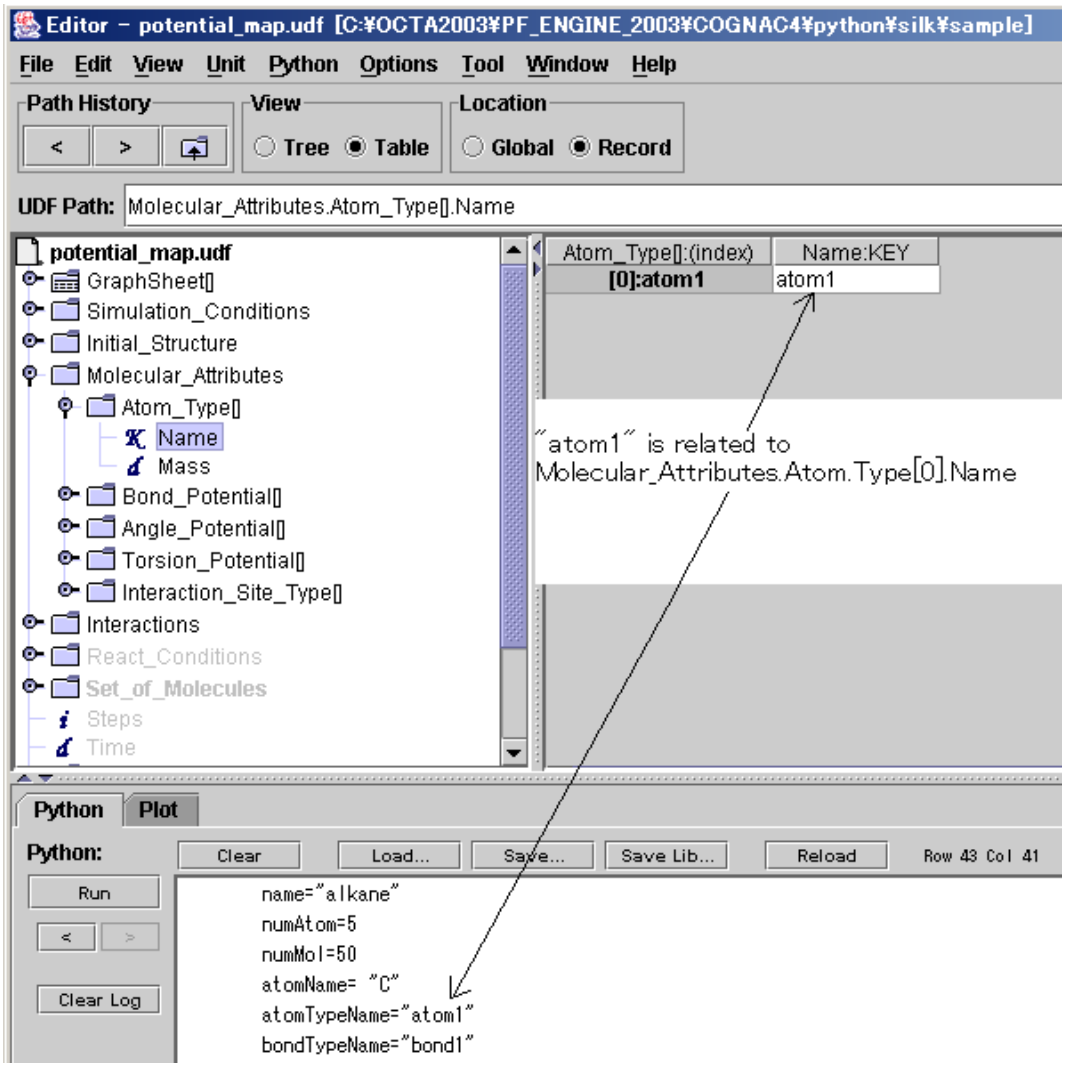


図 4.1: **Atom** のタイプ属性の関連

される（ここでは UDF Path 名 **Molecular_Attributes.Interaction_Site_Type[0]** のデータが参照される）。

図 4.3 に入力 UDF とスクリプトの関連を示す。

(c) **SILK** スクリプトの実行

編集した **SILK** スクリプトの内容（出力先等）を確認し、スクリプトを実行する。実行結果についてはログウインドウに表示される。

4.2.2　入力データの編集と COGNAC の起動

SILK により作成したデータを **GOURMET** に読み込み、データを表示し、内容を確認し、**COGNAC** を起動する。

1. **SILK** により作成した **COGNAC** 入力 UDF の読み込み

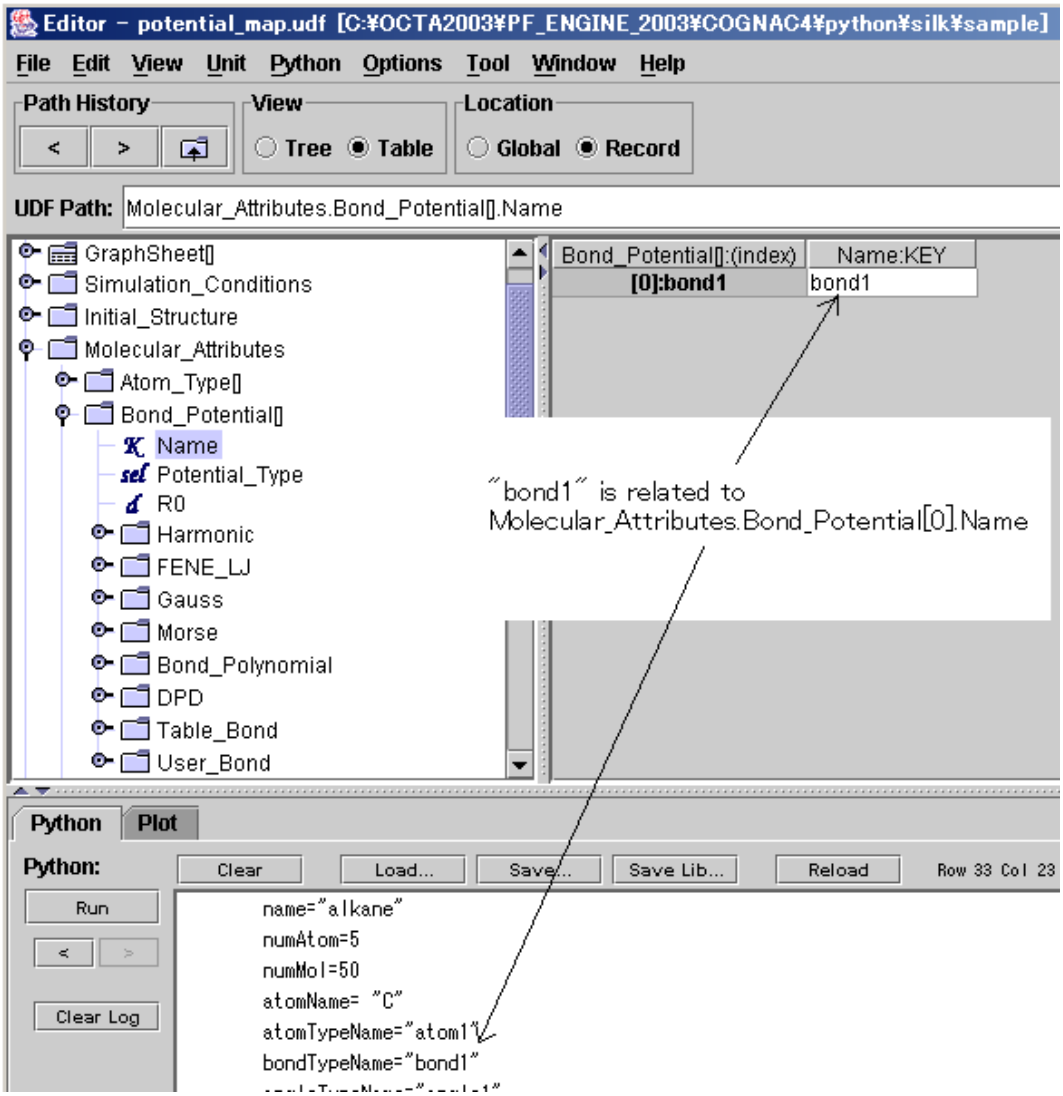


図 4.2: **Bond** のタイプ属性の関連

GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから **SILK** において **self.engine.cognacFileName** で指定した、ファイル名 “pentane50_in.udf” を選択する。

2. ポテンシャルの確認

起動画面の左側にはエクスプローラー様のデータ構造が表示されている。このデータ構造をクリックすることによって現在の Record のデータを参照（編集）することができる。ポテンシャルに関するパラメータは以下のデータ構造に記述されている。

UDF Path 名 **Molecular_Attributes** ··· 分子内ポテンシャル。

UDF Path 名 **Interactions** ··· 分子間ポテンシャルや外場を定義するポテンシャル。

UDF Path 名 **Unit_Parameter** ··· 単位換算を行うために必要な、質量、エネルギー、長さの単位あたりの換算値。

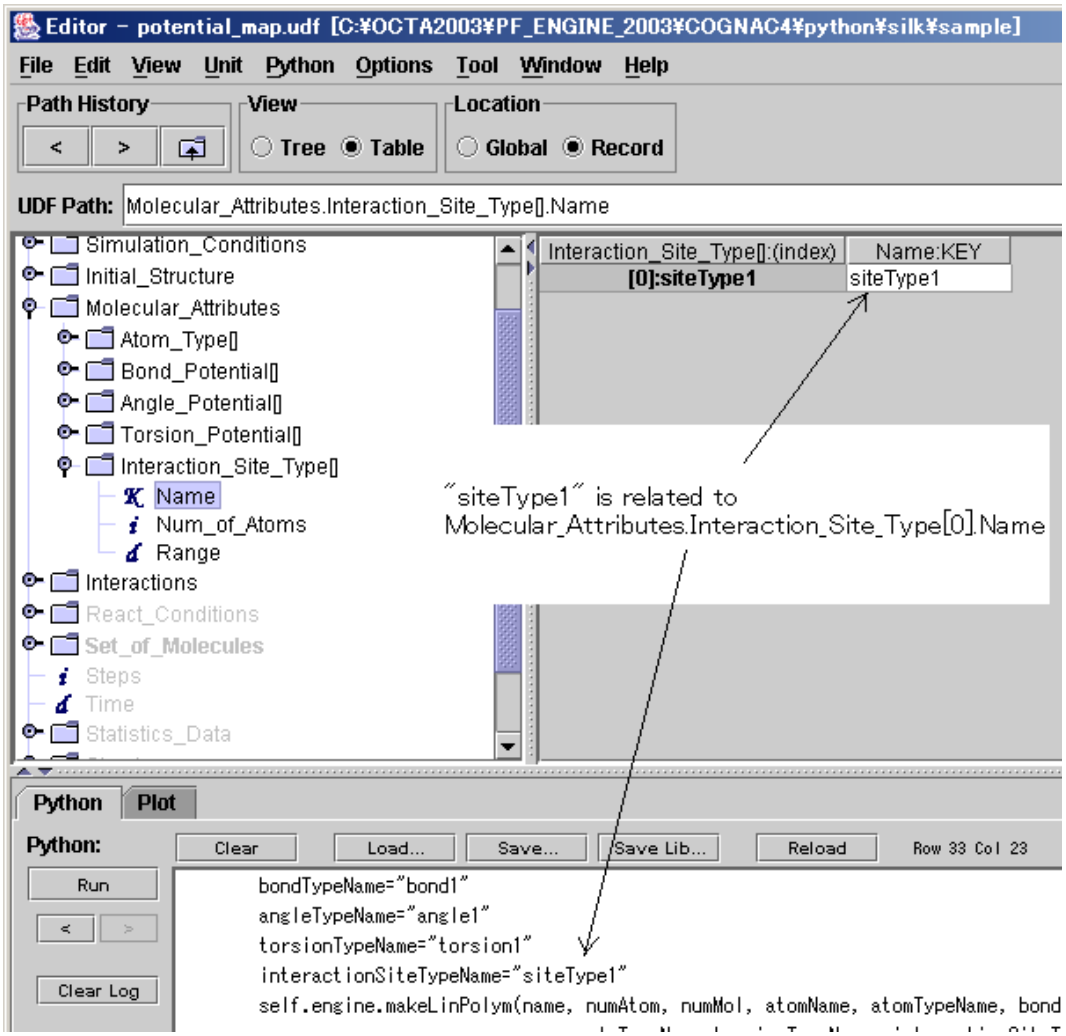


図 4.3: **Interaction_Site** のタイプ属性の関連

Unit_Parameter のデータについて次に記す。

Mass ··· 質量の単位。1 メチレン基の質量 $(= 14amu)$ を 1 単位としている。

Energy ··· エネルギーの単位。分子間相互作用であるレナードジョーンズポテンシャルのパラメータ $\epsilon(= 0.40122kJ/mol)$ を 1 単位としている。

Length ··· 長さの単位。分子間相互作用であるレナードジョーンズポテンシャルのパラメータ $\sigma(= 0.40328nm)$ を 1 単位としている。

これらの単位を決定することにより、温度、圧力等各緒量が計算できる。単位換算のスクリプトとして、“silk_mujigenkun.py” （ “python/silk/silk_mujigenkun.py” ）を Load して使用することができる。

3. 初期構造の設定

UDF Path 名 **Initial_Structure** 以下には、初期構造（各 Atom の座標に関する情報）に関して記述されている。**Initial_Structure** 以下のデータの内、主なものについて説明する。

Initial_Unit_Cell.Density ··· 初期に設定される系の密度。ここには $0.9g/cm^3$ を単位換算して入力している。

Generate_Method.Method ··· 初期構造作成についての方法を記述（予約語）。**Random** と指定。これにより **Generate_Method.Random** のデータが **COGNAC** によって参照される。

同じ階層に存在するその他のデータ（例えばUDF Path 名 **Initial_Structure.Generate_Method.Helix** 等）は無視される。

Relaxation ··· 初期構造の緩和に関するデータを設定する。

通常、分子間相互作用を考慮した状態で初期構造をランダムに発生させた場合は Atom(Molecule) 同士の重なりによって過大な反発力が生じ、シミュレーションが発散してしまう。

それを防ぐために、**Relaxation** のフラグを **1** として初期構造の緩和機能を on にする（off は **0**）。

4. 計算条件の設定

UDF Path 名 **Simulation_Conditions** を編集することにより計算条件を設定する。

設定したパラメータは以下のとおりである。

- ダイナミクス条件

  UDF Path 名 **Simulation_Conditions.Dynamics_Conditions** 以下のデータについて説明する。

  Max_Force ··· シミュレーション中に Atom に許容される最大の力である。ここでは **10000** とする。

  Time.delta_T （設定値:**0.001**）··· 1 ステップ当りの時間。

  Time.Total_Steps （設定値:**2000**）··· シミュレーションの総ステップ数。

  Time.Output_Interval_Steps （設定値:**20**）··· 出力のインターバルステップ。

  Temperature.Temperature （設定値:**3.0**）··· 設定温度。

  Temperature.Interval_of_Scale_Temp（設定値: **10000000**）··· 速度スケーリングのインターバル。

  ***Temperature.Interval_of_Scale_Temp** はシミュレーションの総ステップ数よりも大きいので速度スケーリングは実行されない。ここでは別の手法によって温度制御するので、速度スケーリングの必要は無い。

- ソルバー

  UDF Path 名 ··· **Simulation_Conditions.Solver** 以下のデータについて説明する。

  Solver_Type ··· **Dynamics** 、もしくは **Minimize** のいずれかを選択する。ここではダイナミクスを意味する **Dynamics** を選択する。

  Dynamics.Dynamics_Algorithm ··· 実行するアルゴリズムを選択する。ここでは温度一定のアンサンブル **NVT_Nose_Hoover** を選択する。

  Dynamics.NVT_Nose_Hoover.Q （設定値:**20**）··· **NVT_Nose_Hoover** のアルゴリズムによるダイナミクスに必要なパラメータ。

- 境界条件

  Boundary_Conditions ··· 単位セルの各軸における境界条件を入力する。

  a_axis 、**b_axis** 、**c_axis** を全て **PERIODIC** とすることにより、3 次元周期境界条件が設定される。

- 計算するポテンシャル項の設定

  Calc_Potential_Flags ··· 分子内、分子間（外場を含む）の相互作用の計算条件を入力する。
  各項目について **1** を入力した場合、on となり、**0** を入力した場合 off となる。ここでは結合長（**Bond** ）、結合角（**Angle** ）結合二面角（**Torsion** ）、分子間相互作用（**Non_Bonding** ）を on にする。

- 出力データ項目の設定

  UDF Path 名 ··· **Simulation_Conditions.Output_Flags** 以下のデータについて説明する。

  Statistics ··· 温度や圧力等について何を出力するかについて設定する。
  ここでは全て **1** としている。
  Structure ··· 各 Atom の座標や速度等について何を出力するかについて設定する。
  ここでは座標についてのみ出力するよう、データ名 **Structure.Position** のみ **1** としている。

5. UDF の Save
   各パラメータについて編集操作を施した場合、編集結果が反映されるように **GOURMET** の **File** → **Save** メニューにより UDF data を Save しなければならない。

6. **COGNAC** の起動
   第 3 章　操作入門で示したような手順により **GOURMET Engine Run** コマンドより起動するか、あるいは、コマンドプロンプトあるいはシェルウインドウで、UDF file のあるディレクトリに移動し、以下のコマンドを実行する。

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

   プログラムが終了すると、“pentane50_out.udf”（UDF 出力）、“pentane50_out.dat”（データ出力）、“pentane50.log”（ログファイル）が生成する起動時の引数等入力の詳細に関しては §5.1 を参照。

4.2.3　計算結果の表示と解析

分子構造の表示

計算結果を **GOURMET** に読み込んで、第 3 章　操作入門に示したような手順により、**GOURMET** 画面より、**Action command** を用いて、分子構造を表示することが出来る。
図 4.4 に **type** を **ball-stick** として表示した例を示す。

計算結果データの表示

計算結果を **GOURMET** に読み込んで、データの表示を行う。

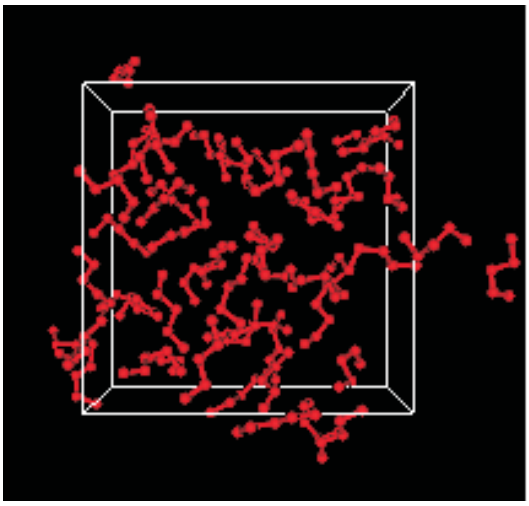


図 4.4: **type = 'ball-stick'** で表示した例

1. 出力 UDF の読み込み

   GOURMET Editor ウインドウより **File → Open...** を選択して、現れるファイルブラウザから “pentane50_out.udf” を選択する。すでに読み込んでいれば、このステップは必要ない。

2. 入力データの表示

   COGNAC の出力 UDF には、計算に用いた入力条件も合わせて出力されるので、入力データを確認することもできる。
   （例）画面左のツリーより **Simulation_Condition → Dynamics_Condition → Time** と選択することにより、シミュレーションの ΔT、トータルステップ、出力のインターバルの入力値が表示される。

3. 出力データの表示

   出力データを表示するためには、**GOURMET** 画面右上の Records を選択し、下部のスライドバー、あるいは右下の表示窓に数値を入力することによりレコード数を指定する。
   （例）Records を選択し、画面左のツリーより **Statistics_Data → Energy → Batch_Average** と選択することにより、エネルギーの区間平均値が各項目に分かれて表示される。

計算結果のグラフ化

計算結果のグラフ化を、温度と圧力の経時変化を例にとり説明する。

1. 出力 UDF の読み込み

   前節と同様に出力 UDF を読み込む。すでに読み込んでいればこのステップは必要ない。

2. 経時変化データの収集

   グラフ作成のため、データの経時 (Record 毎の) 変化を収集する。まず左下の **Python:Load** を選択して、現れるファイルブラウザより “MakeGraphSheet.py” を読み込み **Run** する。その後 **Table mode** において、画面左のツリーの一番上にある、**GraphSheet** を選択すると Record 毎の **Time**、**Temperature**、

Pressure のデータが一つのシートとして作成される。

3. gnuplot のプロット条件の設定

画面左下にある **Python/Plot** のタグから **Plot** を選択してメニューが変わったら **Plot:Make** を選択。窓に gnuplot のコマンドラインが現れる。このままプロットすると、**GraphSheet** の **Index** が横軸になるので以下のように書き換える。

（オリジナル）

```
# 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
```

（書き換え後－温度をプロットする場合）

```
# 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
```

（書き換え後－圧力をプロットする場合）

```
# 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. gnuplot の起動

Plot:Plot を選択することにより gnuplot が立ち上がり、図 4.5 に示すように、温度あるいは圧力の経時変化がプロットされる。

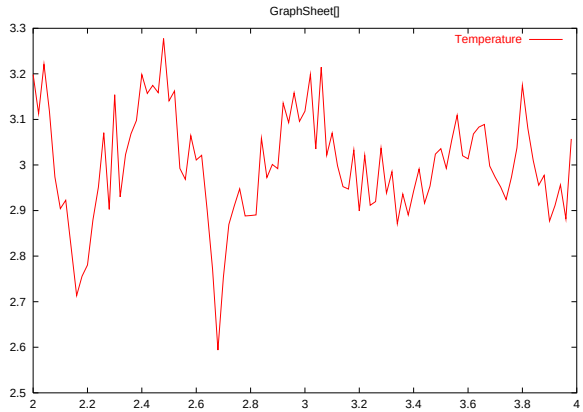


図 4.5: 温度の経時変化プロット例

4.3 （事例 II）ユナイティッドアトムモデルによるポリエチレンオキシド (PEO) のシミュレーション

ここでは、Poly(ethylene oxide)(PEO) の MD シミュレーションの例を示す。特に PEO のように、若干複雑なシーケンスを持つ高分子鎖の **SILK** による構築方法を解説する。

使用するファイルは特に指定するものを除いて、“sample/peo” 以下にある。

4.3.1 SILK による入力データ作成

SILK は Python により記述された、**COGNAC** の入力ファイルを作成する補助ツールである。詳細は第 6 章を参照のこと。

SILK を実行する為には、シミュレーションの条件ならびにポテンシャルが記述された **SILK** 用入力 UDF が必要である。**SILK** 用入力 UDF は、**COGNAC** 入力 UDF とほぼ同じ構造を持つ。雛型として用意されている “potential_map.udf” をとって説明する。

1. **SILK** 入力ファイル（ “potential_map.udf”）

   まず最初に **SILK** 入力ファイル（ “python/silk/sample/potential_map.udf” ）を参照する。ここでは、3 番目のレコード（Record Number は **2**、Record Label は “UA_PEO_LiI”）を使用する。

2. UDF（ “potential_map.udf” ）の Open

   GOURMET を起動し、UDF（ “potential_map.udf”）を Open する。Records モードで表示させると画面には Record Number が **0** のデータが表示される。Open 窓下のつまみを右にスライドさせ、右下の窓に Record Label である “UA_PEO_LiI” が表示されるまで動かす。

3. 分子の作成（パイソンスクリプトの Load と編集ならびに実行）

   ここでは、システムサイズ · · · PEO5 分子と Li イオン 5 個、I イオン 5 個からなる系を作成する手順について説明する。

   (a) パイソンスクリプトの Load

   GOURMET 上で **python/silk/sample/silk_use_peo.py** を Load する。Python 窓に Load されたスクリプトが表示されていることを確認し、スクリプトの上部を参照する。

   (b) Load されたパイソンスクリプトの編集

   Load されたスクリプトの上部にユーザーが編集すべきセクション（**SECTION “USER DEFINITION”**）が記述されている。

   - 出力先

     USER DEFINITION 内の **SUBSECTION “outputpath”** について説明する。

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

self.engine.outDir には出力先のディレクトリを指定する。

self.engine.cognacFileName には出力ファイル名（UDF 拡張子 : .udf を除く）を指定する。この例では “peo_in.udf” という **SILK** の出力、すなわち **COGNAC** 入力ファイルが作成される。

self.engine.prjName には出力ファイルのプロジェクト名を指定する。

self.engine.fileNumCom には出力ファイルを分割する（トポロジーを記述したファイルとその他の条件を記述したファイルに分割する）か否かのコマンドを記述する。
self.engine.fileNumCom=”ONE_FILE” とした場合、単一のファイルにまとめられる。

- 分子の定義（ 1 ）
USER DEFINITION 内の **SUBSECTION “system”** について説明する。ここでは、PEO を作成したスクリプトについて解説する。

```
##### 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)
```

PEO 分子のトポロジーを作製する為に、いくつかの **SILK** が用意した基本的な関数を用いている。Python 文法の詳細についてここでは述べないが、各関数および値について次に説明する。

i. 分子の登録、1 分子あたりの Atom 数の設定

```
numAtom=36
```

1 分子当りの Atom 数である **numAtom** は **36** とする。PEO は $CH_2 - O - CH_2$ を 1 ユニットとし、それが 12 ユニット連なったものとしてモデル化する（分子末端は CH_3）。CH_2（CH_3）は 1Atom としてモデル化されるので、1 分子当り 36 個の Atom からなる。

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

ここでは、分子を登録する関数 **createMolecule(”peo”)** を呼び出している。**self.engine.** は必須の記述である。この関数は **peo** という名前で **SILK** に対して分子を登録したことを意味する。

ii. 登録された分子（分子名: **peo**）への Atom の登録

```
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): は、Python の for 文によってループを回している。**i** という整数を 0 から 36(**numAtom** の値) まで 1 毎に増やしながらその下行を実行する。下行の内容は次の通り。
i を 3 で割った剰余が 1 の時、**SILK** の関数 **addAtoms("peo", "O", "atom2")** がコールされる。
この剰余が 1 でない時、**SILK** の関数 **addAtoms("peo", "C", "atom1")** がコールされる。
ここで剰余が 1 の時とは、$CH_3-O-CH_2-CH_2-O-CH_2-CH_2-O-CH_2\cdots$ の O 原子を指す。また、剰余が 1 でない時とは、CH_2（CH_3）を指す。
すなわち、**SILK** に対して登録した分子"peo"について PEO 分子内の Atom シーケンスにしたがって Atom を登録している。

iii. 登録された分子（分子名:**peo**）への Bond の登録

```
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−1**):** は、Python の for 文によってループを回している。**i** という整数を 0 から 35(**numAtom** の値 −1) まで 1 毎に増やしながらその下行を実行する。ここでは Bond を登録するので、その数が **numAtom**−1 となる。下行の内容は次の通り。
i を 3 で割った剰余が 2 の時、**SILK** の関数 **addBonds("peo", i, i+1, "bond2")** がコールされる。
この剰余が 2 でない時、**SILK** の関数 **addBonds("peo", i, i+1, "bond1")** がコールされる。
ここで剰余が 2 の時とは、$CH_3-O-CH_2-CH_2-O-CH_2-CH_2-O-CH_2\cdots$ の CH_2-CH_2 間（あるいは CH_3-CH_2）の Bond を指す。
また、剰余が 2 でない時とは、CH_2-O 間（あるいは CH_3-O）の Bond を指す。
すなわち、**SILK** に対して登録した分子 **peo** について PEO 分子内の Atom シーケンスにしたがって Bond を登録している。
関数で呼ばれている引数 **bond1**、もしくは **bond2** は、対応する **BondType** の名前 **Name** を設定する。すなわち **bond1** という名前を持つ **Bond_Potential** のデータ（この場合、UDF Path 名 **Molecular_Attributes.Bond_Potential[0]**）が参照される。

iv. 登録された分子（分子名:**peo**）への Angle、Torsion の登録

```
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")
```

Angle、Torsion については、Bond と同じ方法で登録できる。Angle を登録する関数として **addAngles(...)** が、また Torsion を登録する関数として **addTorsions(...)** が用意されている。

v. 登録された分子（分子名:**peo**）への Interaction Site の登録

```
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")
```

ここでは分子間相互作用を作用させる **Interaction Site** を登録している。**SILK** が用意する関数は **addInteractionSites(...)** である。引数として呼ばれている [**i**] は、Atom の配列を示す。これは **COGNAC** の特徴である複数の Atom から定義される Interaction Site をサポートする機能に対応したものである。また、第 4 引数 **"PAIR"** は登録する Interaction Site が Pair interaction に由来するものであることを指している。

vi. 登録された分子（分子名:**peo**）への Electrostatic Site の登録

```
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)
```

ここでは静電相互作用を作用させる Electrostatic Site を登録している。**SILK** が用意する関数は **addInteractionSites(...)** である。第 4 引数 **"COULOMB"** が、静電相互作用を定義するサイトであることを宣言している。また、第 3 引数 **"POINT_CHARGE"** という名前を持つ **Electrostatic_Interaction** のデータ
（ UDF Path 名 **Interactions.Electrostatic_Interaction**[**0**]）が参照される。
第 5 引数は双極子能率もしくは電荷が設定される。

vii. 登録された分子（分子名:**peo**）の内、出力する分子を **SILK** へ登録

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

ここでは登録した PEO 分子（名前:**peo**）について、何本の分子を出力するかを指定する。**SILK** に分子を登録、作製しても、この関数 **setSystem(. . .)** が呼ばれない限り出力はされないので注意。

- 分子の定義（2）
  USER DEFINITION 内の **SUBSECTION “system”** について説明する。ここでは、Li イオンおよび I イオンを作成したスクリプトについて次に記す。Li イオンおよび I イオンについては単原子分子とみなすことができるので、以下のようにして記述する。

  *Li イオン 5 個の登録

```
#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)
```

  *I イオン 5 個の登録

```
#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) パイソンスクリプトの実行
編集したパイソンスクリプトの内容（出力先等）を確認し、スクリプトを実行する。実行結果についてはログウインドウに表示される。

4.3.2 入力データの編集と COGNAC の起動

SILK により作成したデータを **GOURMET** に読み込み、データの編集を行ってエンジンを起動する。

1. **SILK** により作成した **COGNAC** 入力 UDF の読み込み
   GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから **SILK** において **self.engine.cognacFileName** で指定した、ファイル名 “peo_in.udf” を選択する。

2. ポテンシャルの確認

起動画面の左側にあるエクスプローラー様のデータ構造の各項目をクリックすることによってデータを参照（編集）する。

UDF Path 名 “Unit_Parameter” には単位換算を行うために必要な、質量、エネルギー、長さの単位あたりの換算値が入力されている。このポテンシャルは Jorgensen らが開発した PEO のユナイテッドアトムモデルのポテンシャルを元にしたものである [54, 55]。**Unit_Parameter** のデータについて次に記す。

Mass_Unit__amu ··· 質量の単位。$10amu$ を 1 単位としている。

Energy ··· エネルギーの単位。$1kcal/mol(4.18605kJ/mol)$ を 1 単位としている。

Length ··· 長さの単位。$1Å(0.1nm)$ を 1 単位としている。

これらの単位を決定することにより、温度、圧力等各緒量が計算できる。単位換算のスクリプトととして、“silk_mujigenkun.py”（ “python/silk/silk_mujigenkun.py”）を Load して使用することができる。先に設定した電荷の値もこのスクリプトを用いて、単位換算を行うことが出来る。

3. 初期構造の設定

   UDF Path 名 **Initial_Structure** で初期構造を設定する。ここではリスタートを選択する（リスタート UDF は “peo_rst.udf”）。

4. 計算条件の設定

   UDF Path 名 **Simulation_Conditions** を編集することにより計算条件を設定する。設定したパラメータは以下のとおりである。

   (a) ダイナミクス条件

   UDF Path 名 ··· **Simulation_Conditions.Dynamics_Conditions** 以下の主なデータについて説明する。

   Time.delta_T（設定値:**0.01617**）··· 1 ステップ当りの時間（ここでは 2.5fs）。
   Time.Total_Steps（設定値:**100,000**）··· シミュレーションの総ステップ数である。
   Time.Output_Interval_Steps（設定値:**200**）··· 出力のインターバルを示す。

   Li イオンの拡散挙動を解析するために、シミュレーションタイムステップを変更して長時間のシミュレーションを行う。**Simulation_Conditions.Dynamics_Conditions.Time** を開き、図 4.6 に示す様に **Total_Steps** を **100,000** 程度に変更する。同時に精度を高めるために、**Output_Interval_Steps** を **200** にしてサンプル数を増やす。

   Temperature.Temperature（設定値:**0.715**）··· 設定温度（ここでは約 360K）。
   Temperature.Interval_of_Scale_Temp（設定値:**0**）··· 速度スケーリングのインターバルステップ数。

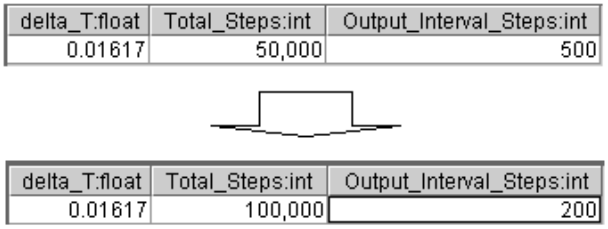


図 4.6: **Total_Steps** の変更

*__Temperature.Interval_of_Scale_Temp__ を **0** にした場合、速度スケーリングは実行されない。ここでは別の手法によって温度制御するので（速度スケーリングの）必要は無い。

(b) ソルバー

UDF Path 名 · · · **Simulation_Conditions.Solver** 以下の主なデータについて説明する。

Solver_Type（設定値:**Dyanimcs**）· · · ダイナミクス。

Dynamics.Dyanamics_Algorithm（設定値: **NVT_Nose_Hoover**）· · · 実行するアルゴリズム（ここでは Nose-Hoover の手法による NVT アルゴリズムを使用）。

Dynamics.NVT_Nose_Hoover.Q（設定値:**20**）· · · **NVT_Nose_Hoover** のアルゴリズムによるダイナミクスに必要なパラメータ。

(c) 境界条件

Boundary_Conditions（設定値:**PERIODIC** （**a_axis**、**b_axis**、**c_axis** 全て））· · · 境界条件の設定

(d) 計算するポテンシャル項の設定

Calc_Potential_Flags.Bond（設定値：**1**）

Calc_Potential_Flags.Angle（設定値：**1**）

Calc_Potential_Flags.Torsion（設定値：**1**）· · · 分子内相互作用の計算条件。

Calc_Potential_Flags.Non_Bonding（設定値：**1**）

Calc_Potential_Flags.Electrostatic（設定値：**1**）· · · 分子間相互作用の計算条件。

(e) 出力データ項目の設定

UDF Path 名 · · · **Simulation_Conditions.Output_Flags** 以下の主なデータについて説明する。

Statistics（設定値：全て **1**） · · · 温度や圧力等の出力項目の選択。（ここでは全ての項目について出力する）

Structure.Position（設定値：**1**）

Structure.Velocity（設定値：**0**）

Structure.Force（設定値：**0**）· · · 各 Atom の座標、速度、力について出力の選択。座標のみ出力する。

5. UDF の Save

   各パラメータについて編集操作を施した場合、編集結果が反映されるように **GOURMET** のメニューから Save をしなければならない。

6. **COGNAC** の起動

   第 3 章　操作入門で示したような手順により **GOURMET Engine Run** コマンドより起動するか、あるいは、コマンドプロンプトあるいはシェルウインドウで、UDF を保存したディレクトリに移動し以下のコマンドを実行する。

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

   プログラムが終了すると、“peo_out.udf”（UDF 出力）、“peo_out.dat”（データ出力）、“peo.log”（ログファイル）が生成する

4.3.3　計算結果の表示と解析

分子構造の表示

計算結果を **GOURMET** に読み込んで、第 3 章　操作入門に示したような手順により、**GOURMET** 画面より、**Action command** を用いて、分子構造を表示することが出来る。

平均自乗変位 (MSD) の解析

Li イオンの拡散挙動を解析するために平均自乗変位 (MSD) を計算する方法を以下に解説する。同様の解析は **Action command : ANALYSIS_msd...** を用いて行うことも出来る。詳細は §7.5 参照。

1. 出力 UDF の読み込み

   GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから “peo_out.udf” を選択する。

2. Python script の読み込みと実行

   GOURMET 画面の左下の **Python:Load** を選択して現れるファイルブラウザから”msd.py” を選択して読み込む。つづいて Run すると **Table mode** において現れる **GraphSheet** に Li 原子の MSD および MSD の各軸成分が時間の関数としてセットされる。

3. MSD のプロット

   GraphSheet を選択した状態で画面左下にある **Python/Plot** のタグから **Plot** を選択してメニューが変わったら **Plot:Make** を選択。窓に gnuplot のコマンドラインが現れる。このままプロットすると、**GraphSheet** の **Index** が横軸になるので以下のように書き換える

   （オリジナル）

   ```
   # 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
```

（書き換え）

```
# 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
```

その後 **Plot:Plot** を選択すると gnuplot が立ち上がり図 4.7 に示すように、MSD および MSD の各軸 (x,y,z) 成分がプロットされる。

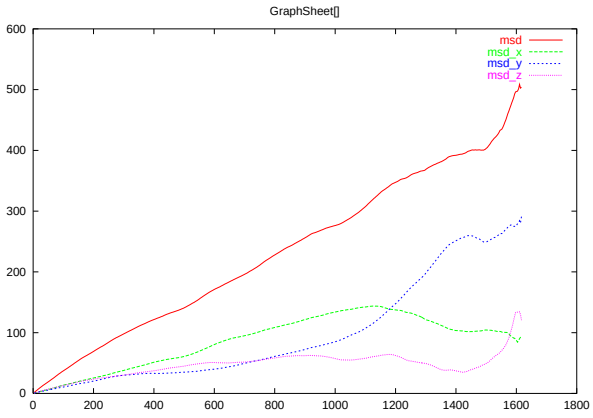


図 4.7: 平均二乗変位のプロット例

実際に拡散定数などを求める場合、時間軸の後半のデータはデータ点数が少なく、精度的に問題があるので、前半部分を用いて解析するとよい。

4.4 （事例 III）SUSHI からのズームインを利用した ABA トリブロックコポリマーラメラのシミュレーション

ここでは、**SUSHI** により計算された出力を元にして、ブロックコポリマーのラメラ構造を作成する方法を説明する。ポリマーモデルは操作入門で解説したものと同じものであるので、**SILK** によるデータ作成に関してはそちらを参照のこと。

使用するファイルは特に指定するものを除いて、“sample/blend” 以下にある。

また **SUSHI - COGNAC** のズーミングに関する事例は適用研究事例 AMUSE ドキュメントにも紹介されているのでそちらも参照のこと。

4.4.1 SUSHI の計算条件

SUSHI の出力ファイルより、計算条件をチェックすることができる。詳細は「SUSHI ユーザーズ マニュアル」を参照のこと。**COGNAC** で用いるための注意点として、**segment_volume_fraction** をすべてのセグメント毎に出力しておく必要がある。その他は通常の **SUSHI** の計算条件と変わらない。
SUSHI の入出力に関しては「SUSHI ユーザーズ マニュアル」参照のこと。

4.4.2 入力データの編集と COGNAC の起動

ここでは第 3 章　操作入門で用いた入力 UDF を編集して、**SUSHI** からのズームインを利用する入力 UDF を作る方法を説明する。サンプルには出来上がったファイル “A20B40A20_zoom_in.udf” もあるのでそちらも参照のこと。

1. 入力 UDF の読み込み

   GOURMET Editor ウインドウより、**File** → **Open...** を選択して、現れるファイルブラウザから “A20B40A20_in.udf” を選択する。

2. **External_Interaction** の追加

   Density biased potential を加えるために、**External_Interaction** の設定を行う。
   まず **Interactions.External_Interaction[]** を開くと図 4.8 に示すように、配列要素の無いデータが表示される。

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

図 4.8: **External_Interaction[]** の初期状態

この空の列を指定して、**Edit→Insert Row** を 2 度行い、**External_Interaction** の要素を 2 つ作成する。
そして各要素に図 4.9 に示すようにデータを入力して、外場の設定を行う **Interaction_Site** を指定する。

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

図 4.9: **External_Interaction** の入力

次に、**Interactions.External_Interaction[].Density_Field** を開き、図 4.10 に示すように、外場のタイプとして **Density_Biased_Potential** を選択すると同時に、外場のベースとなる **SUSHI** の出力

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	{

図 4.10: **Density_Field** の入力

UDF とセグメント種を **Component_Index** で指定する。

さらに、**Interactions.External_Interaction[].Density_Field.Density_Biased_Potential** において相互作用パラメータ **chi** を指定する。

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

図 4.11: **Density_Biased_Potential** の入力

ここで 2 行目において **chi** が **SUSHI** で用いている χ parameter の 2 倍に設定されているのは、ポテンシャルを計算するベースとなる **Component_Index = 0** の体積分率が、対称 ABA トリブロック子ポリマーであるため、セグメント **A** のトータルの体積分率の 1/2 になっているからであることに注意。

3. **Initial_Unit_Cell** の変更

   SUSHI で計算したラメラ周期とセルサイズにあわせて、**COGNAC** のセルサイズも変更する。
   Initial_Structure.Initial_Unit_Cell.Cell_Size を開いて **c** に **SUSHI** で計算したメッシュサイズと同じ **39.0** と入力する。これにより **c=39.0**、**a** および **b** は密度より算出されるサイズにセットされる。

4. **Node_Density_Bias** の追加

   初期構造を発生する際に **SUSHI** で計算した segment density を利用するための設定をおこなう。
   Initial_Structure.Generate_Method.Random.Node_Density_Bias[] を開き、**External_Interaction** の設定の際と同様に **Edit→Insert Row** を行い、入力行を作る。
   次に図 4.12 に示すように各項に入力して、**SUSHI** の segment density の **0-79** を分子鎖 **A20B40A20** に端から対応させる。

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

図 4.12: **Node_Density_Bias** の入力

5. データの保存

   File → Save As を選択して編集した UDF を別名（例えば “A20B40A20_zoom_in.udf”）で保存する

6. **COGNAC** の起動

   第 3 章で示したような手順により **GOURMET** Engine Run コマンドより起動するか、あるいは、コマンドプロンプトあるいはシェルウインドウで、UDF を保存したディレクトリに移動し以下のコマンドを実行する。

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

   入力 UDF の名前は、保存の際につけたファイル名にする。プログラムが終了すると、
   “A20B40A20_zoom_out.udf”（UDF 出力）、“A20B40A20_zoom_out.dat”（データ出力）、
   “A20B40A20_zoom.log”（ログファイル）が生成する

   GOURMET Engine Run コマンドより起動する場合、用いる **SUSHI** の UDF file を **(Local input Files)** の **Parms UDF:**で指定するとよい。

4.4.3 計算結果の表示と解析

分子構造の表示

計算結果を **GOURMET** に読み込んで、第 3 章　操作入門に示したような手順により、**GOURMET** 画面より、**Action command** を用いて、分子構造を表示することが出来る。

動径分布の解析

第 3 章　操作入門で説明したのと同様に、**GOURMET** で読み込んで動径分布 $g_{AB}(r)$ を計算し、プロットすることができる。ラメラ構造を取っているためにランダムな構造の結果と異なることを観察する。

濃度分布の解析

1 次元方向の体積分率の分布を計算し、プロットする。同様の解析は **Action command : ANALYSIS_1D_profile...** を用いて行うことも出来る。詳細は §7.5 参照。

1. 出力 UDF の読み込み

   前節と同様に出力 UDF を読み込む。すでに読み込んでいればこのステップは必要ない。

2. Record の移動

   画面下のスライドバーで最後のレコード (**11**) まで移動する

3. Python script の読み込みと実行

   GOURMET 画面の左下の **Python:Load** を選択して現れるファイルブラウザから “density1D.py” を選択して読み込む。つづいて Run すると **Table mode** において現れる **GraphSheet** に Z、$\phi_A(Z)$、$\phi_B(Z)$ のデータがセットされる。

4. $\phi_A(Z)$、$\phi_B(Z)$ のプロット

   GraphSheet を選択した状態で画面左下にある **Python/Plot** のタグから **Plot** を選択してメニューが変わったら **Plot:Make** を選択。窓に gnuplot のコマンドラインが現れる。このままプロットすると、**GraphSheet** の **Index** が横軸になるので以下のように書き換える。

   （オリジナル）

```
# 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
```

   （書き換え）

```
# 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
```

   その後 **Plot:Plot** を選択すると gnuplot が立ち上がり図 4.13 に示すように、体積分率 $\phi_A(Z)$、$\phi_B(Z)$ がプロットされる。

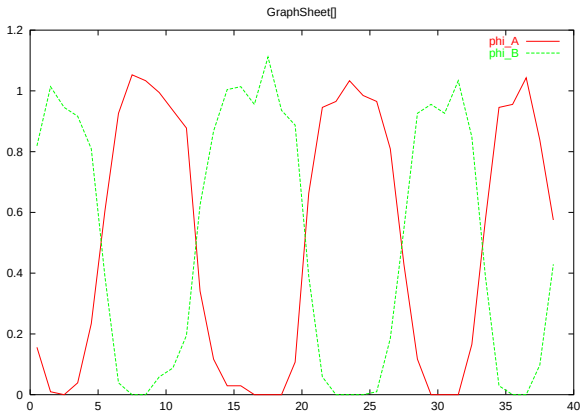


図 4.13: 体積分率のプロット例

4.5　（事例 IV）Table を用いたポテンシャルによるシミュレーション

相互作用ポテンシャルに数値テーブルデータを用いる例を示す。モデル、ポテンシャルは、§4.2 で紹介した pentane と同一のものであるので、**SILK** によるデータ作成に関してはそちらを参照されたい。

使用するファイルは特に指定するものを除いて、“sample/tablepotential” 以下にある。

4.5.1 Potential table の確認

Bond potential table を読み込んで内容を確認する

1. “bond_table.udf” の読み込み

   GOURMET Editor/Browser ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから “bond_table.udf” 選択する。

2. 最小、最大値、分割数の確認

   図 4.14 に示すように、**Mesh.axes[].values[]** を開き、以下のデータを確認する。
   value[0] ··· 最小距離
   value[1] ··· 最大距離
   value[2] ··· 分割数

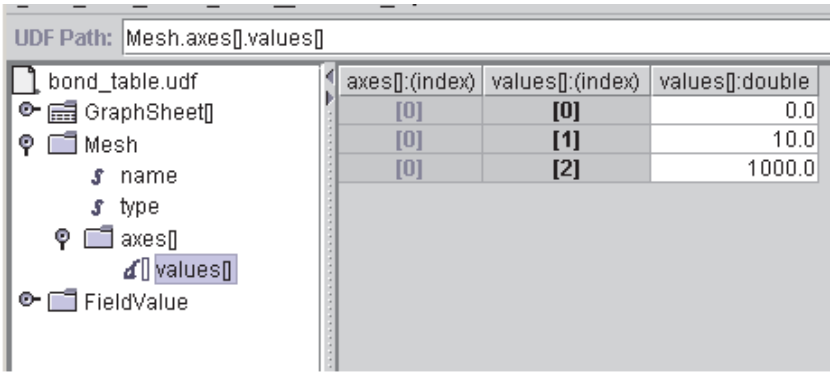


図 4.14: **Bond_Potential_Table** の例

3. ポテンシャルの確認

   FieldValue.value[] を開きデータを確認する。最小距離 → 最大距離に対する、ポテンシャルエネルギーの値がセットされている。

4.5.2 入力データの編集と COGNAC の起動

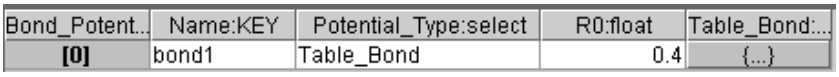
ここでは §4.2 で用いた入力 UDF を編集して入力 UDF を作る方法を説明する。サンプルには出来上がったファイル “pentane_table_in.udf” もあるのでそちらも参照のこと。

1. **COGNAC** 入力 UDF の読み込み

   GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから “pentaneNVT_in.udf” を選択する。

2. Potential type の変更と table potential file の指定

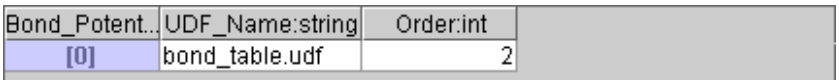
   Bond_Potential を例にとる。**Molecular_Attributes.Bond_Potential[]** を開き、現れた **Harmonic** potential 指定の部分を以下のように変更する。



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

図 4.15: **Bond_Potential** の変更

次に、**Molecular_Attributes.Bond_Potential**[]**.Table_Bond** を開き以下のように Potential table file name、内挿次数を指定する。



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

図 4.16: **Table_Bond** の設定

Angle_Potential[]、**Torsion_Potential**[]、**Interactions.Pair_Interaction**[] も同様に設定する。

3. データの保存

   File → **Save As** を選択して編集した UDF を別名で保存する

4. **COGNAC** の起動

   コマンドプロンプトあるいはシェルウインドウで、UDF を保存したディレクトリに移動し以下のコマンドを実行する。

```
cognac92 -I pentane_table_in.udf -O pentane_table_out.udf > pentane_table.log
```

   入力 UDF の名前は、保存の際につけたファイル名にする。プログラムが終了すると、“pentane_table_out.udf”（UDF 出力）、“pentane_table_out.dat”（データ出力）、“pentane_table.log”（ログファイル）が生成する。

4.5.3　計算結果の表示と解析

分子構造の表示

計算結果を **GOURMET** に読み込んで、第 3 章　操作入門に示したような手順により、**GOURMET** 画面より、**Action command** を用いて、分子構造を表示することが出来る。

計算結果のグラフ化

§4.2 で説明したのと同様に、Temperature、Pressure の経時変化をプロットし、結果を比較してほぼ等しく結果が得られていることを確認する。

4.6 （事例 V）化学反応を取り入れたシミュレーション

ここでは、化学反応を模した結合の生成を考慮した MD シミュレーションの例を示す。
使用するファイルは特に指定するものを除いて、“sample/reaction” 以下にある。

4.6.1 SILK による入力データ作成

1. **SILK** 入力ファイル（ “potential_map.udf” ）

   まず最初に **SILK** 入力ファイル（ “python/silk/sample/potential_map.udf”）を参照する。ここでは、6 番目のレコード（Record Number は **5**、Record Label は “BS_REACTION”）を使用する。

2. UDF（ “potential_map.udf” ）の Open

   GOURMET を起動し、UDF（ “potential_map.udf”）を Open する。Open 時、画面には Record Numebr が **0** のデータが表示されているので Open 窓下のつまみを右にスライドさせ、右下の窓に Record Label である “BS_REACTION” が表示されるまで動かす。

3. 分子の作成（**SILK** スクリプトの Load と編集ならびに実行）

   ここでは、2 種類の単原子分子 100 個からなる系を作成する手順について説明する。

   (a) **SILK** スクリプトの Load

   GOURMET 上で “silk/bin/silk_use_reaction.py” を Load する。Python 窓に Load されたスクリプトが表示されていることを確認し、スクリプトの上部を参照する。

   (b) Load された **SILK** スクリプトの編集

   Load されたスクリプトの上部にユーザーが編集すべきセクション（**SECTION “USER DEFINITION”**）が記述されている。

   - 出力先

     “USER DEFINITION” 内の **SUBSECTION “outputpath”** について説明する。

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

self.engine.outDir には出力先のディレクトリを指定する。

self.engine.cognacFileName には出力ファイル名（UDF 拡張子：.udf を除く）を指定する。この例では “react3_in.udf” という **SILK** の出力、すなわち **COGNAC** 入力ファイルが作成される。

self.engine.prjName には出力ファイルのプロジェクト名を指定する。

self.engine.fileNumCom には出力ファイルを分割する（トポロジーを記述したファイルとその他の条件を記述したファイルに分割する）か否かのコマンドを記述する。

self.engine.fileNumCom=”ONE_FILE” とした場合、単一のファイルにまとめられる。

- 分子の作成（ 1 ）
  “USER DEFINITION” 内の **SUBSECTION “system”** について説明する。

```
##### 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)
#
```

各関数および値について次に説明する。

i. 分子名の登録

self.engine.createMolecule(”beadC”)

”beadC”は分子に設定される名前である。

ii. 登録された分子（分子名:**beadC**）への Atom の登録

self.engine.addAtoms(”beadC”, ”C”, ”C”)

SILK に対して登録した分子 **beadC** について Atom を登録している。
分子 **beadC** は単原子分子としてモデル化されるので、この関数は一度（Atom 一個分）実行される。

iii. 登録された分子（分子名:**beadC**）への Interaction site の登録

self.engine.addInteractionSites(”beadC”, [0], ”siteType1”, ”PAIR”)

ここでは分子間相互作用を作用させる Interaction site を登録している。
引数として呼ばれている [**0**] は、Atom の配列を示す。また、**SILK** は **siteType1_siteType1** という名前を持つ Pair interaction のデータとして、配列 **Interaction.Pair_Interactions**[] に含まれるデータ（この場合 **Interactions.Pair_Interaction**[**0**]）を参照する。

iv. 登録された分子（分子名:**beadC**）の内、出力する分子を **SILK** へ登録

self.engine.setSystem(”beadC”, 60)

ここでは登録した分子（名前:**beadC**）について、何個の分子を出力するかを指定する。**SILK** に分子を登録、作製しても、この関数 **setSystem(. . .)** が呼ばれない限り出力はされないので注意。

- 分子の作成（２）

```
#
self.engine.createMolecule("beadC2")
self.engine.addAtoms("beadC2", "C2", "C2")
self.engine.addInteractionSites("beadC2", [0], "siteType1", "PAIR")
self.engine.setSystem("beadC2", 40)
```

ここでは、**beadC2** という名前の単原子分子を作成している。使用した関数等は上の **beadC** 分子と全く同じなので省略する。

(c) パイソンスクリプトの実行

編集したパイソンスクリプトの内容（出力先等）を確認し、スクリプトを実行する。実行結果についてはログウインドウに表示される。

4.6.2 入力データの編集と COGNAC の起動

SILK により作成したデータを **GOURMET** に読み込み、データを表示し、内容を確認し、エンジンを起動する。

1. **SILK** により作成した **COGNAC** 入力 UDF の読み込み

   GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから **SILK** において **self.engine.cognacFileName** で指定した、ファイル名 “react3_in.udf” を選択する。

2. ポテンシャルの確認

   起動画面の左側にはエクスプローラー様のデータ構造の各項目をクリックすることによってデータを参照（編集）する。

   UDF Path 名 **Molecular_Attributes.Bond_Potential**[] には、**COGNAC** シミュレーション中に反応によって付加される Bond ポテンシャルが登録されている。

*ここに収録されたサンプルとして収録されたデータについては、特定種の分子を対象にしたものではないので、先のセクションで述べた UDF Path 名 **Unit_Parameter** のデータは入力されてない。

3. 初期構造の設定

UDF Path 名 **Initial_Structure** で初期構造設定する。ここでは密度:**0.6** としてランダムに構造を発生させる。

4. 計算条件の設定

UDF Path 名 **Simulation_Conditions** を編集することにより計算条件を設定する。設定したパラメータは以下のとおりである。

(a) ダイナミクス条件

UDF Path 名 ··· **Simulation_Conditions.Dynamics_Conditions** 以下のデータについて説明する。

Time.delta_T（設定値:**0.001**）··· 1 ステップ当りの時間。
Time.Total_Steps（設定値:**10000**）··· シミュレーションの総ステップ数である。
Time.Output_Interval_Steps（設定値:**20**）··· 出力のインターバルを示す。
Temperature.Temperature（設定値:**5.0**）··· 設定温度。
Temperature.Interval_of_Scale_Temp（設定値:**1000000**）··· 速度スケーリングのインターバルステップ数。

***Temperature.Interval_of_Scale_Temp** はシミュレーションの総ステップ数よりも大きいので速度スケーリングは実行されない。ここでは別の手法によって温度制御するので、速度スケーリングの必要は無い。

(b) ソルバー

UDF Path 名 ··· **Simulation_Conditions.Solver** 以下のデータについて説明する。

Solver_Type（設定値:**Dyanimcs**）··· ダイナミクス。
Dynamics.Dyanamics_Algorithm（設定値: **NVT_Kremer_Grest**）··· 実行するアルゴリズム。
Dynamics.NVT_Kremer_Grest.Friction（設定値:**0.5**）··· **NVT_Kremer_Grest** のアルゴリズムによるダイナミクスに必要なパラメータ。

(c) 境界条件

Boundary_Conditions（設定値:**PERIODIC**（**a_axis**、**b_axis**、**c_axis** 全て））··· 境界条件の設定。

(d) 計算するポテンシャル項の設定

Calc_Potential_Flags.Bond（設定値：**1**）
Calc_Potential_Flags.Angle（設定値：**0**）

Calc_Potential_Flags.Torsion（設定値：**0**）··· 分子内相互作用の計算条件。

Calc_Potential_Flags.Non_Bonding（設定値：**1**）··· 分子間相互作用の計算条件。

(e) 出力データ項目の設定
UDF Path 名 ··· **Simulation_Conditions.Output_Flags** 以下のデータについて説明する。

Structure.Position（設定値：**1**）
Structure.Velocity（設定値：**1**）
Structure.Force（設定値：**0**）··· 各 Atom の座標、速度、力について出力の選択。座標、速度について出力する。

5. UDF の Save

各パラメータについて編集操作を施した場合、編集結果が反映されるように **GOURMET** のメニューから Save をしなければならない。

6. **COGNAC** の起動

第 3 章 操作入門で示したような手順により **GOURMET** Engine Run コマンドより起動するか、あるいは、コマンドプロンプトあるいはシェルウインドウで、UDF を保存したディレクトリに移動し以下のコマンドを実行する。

```
cognac92 -I react3_in.udf -O react3_out.udf > react3.log
```

プログラムが終了すると、“react3_out.udf”（UDF 出力）、“react3_out.dat”（データ出力）、“react3.log”（ログファイル）が生成する

4.6.3 計算結果の表示と解析

分子構造の表示

計算結果を **GOURMET** に読み込んで、第 3 章　操作入門に示したような手順により、**GOURMET** 画面より、**Action command** を用いて、分子構造を表示することが出来る。
その際、boundary condition の設定を変えた場合、表示される原子間の距離が、実際の結合距離に対応する結合のみが表示される。実際に結合が存在する場合でも、表示されている座標に基づいて結合を描いた場合、境界条件の関係で実際の結合を反映しない場合は表示しないので、表示上は実際に生成している結合より、結合の数が少なめに現れるので注意。

分子数変化の解析

反応の進行による分子数の減少を解析する。

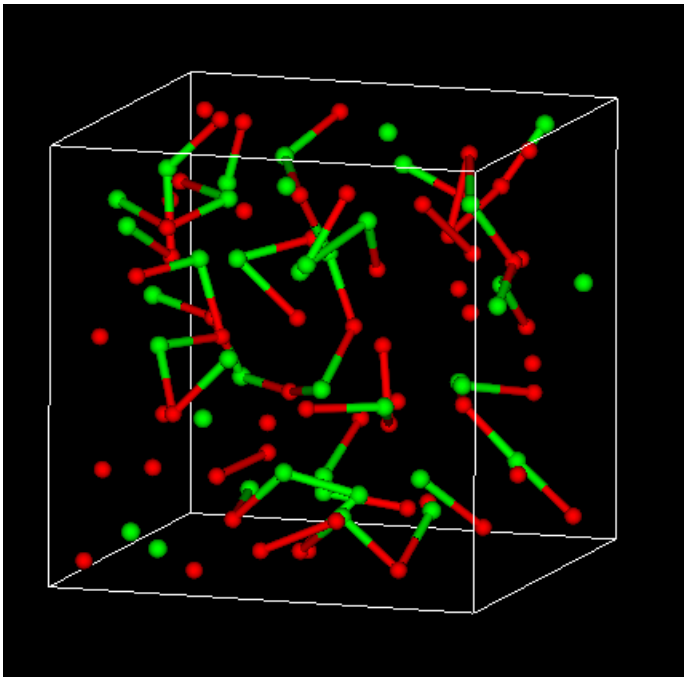


図 4.17: **boundary_condition='atom'** で表示した例

1. 出力 UDF の読み込み

   GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから “reaction_out.udf” を選択する。

2. Python script の読み込みと実行

   GOURMET 画面の左下の **Python:Load** を選択して現れるファイルブラウザから “CounyMol.py” を選択して読み込む。つづいて Run すると、**Table mode** において現れる **GraphSheet** に分子数の経時変化を表したシートが生成する

3. gnuplot のプロット条件の設定

   画面左下にある **Python/Plot** のタグから **Plot** を選択してメニューが変わったら **Plot:Make** を選択。窓に gnuplot のコマンドラインが現れる。このままプロットすると、**GraphSheet** の **Index** が横軸になるので以下のように書き換える。

   （オリジナル）

```
# 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
```

   （書き換え後）

```
# 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
```

4. gnuplot の起動

   Plot:Plot を選択することにより gnuplot が立ち上がり、図 4.18 に示すように、分子数の経時変化がプロットされる。

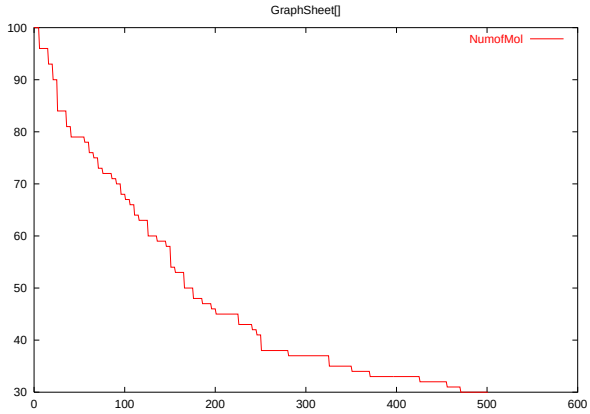


図 4.18: 分子数の経時変化

4.7 （事例 VI）半結晶ラメラシミュレーション

Lamella generator により、平均場理論より求められるラメラ間非晶のコンフォメーションを再現するようなラメラ構造を構築する。
使用するファイルは特に指定するものを除いて、“sample/lamella” 以下にある。

4.7.1 SILK による入力データ作成

1. **SILK** 入力ファイル（ “potential_map.udf” ）

   まず最初に **SILK** 入力ファイル（ “python/silk/sample/potential_map.udf” ）を参照する。ここでは、5 番目のレコード（Record Number は **4**、Record Label は “BS_LAMELLA”）を使用する。

2. UDF（ “potential_map.udf” ）の Open

   GOURMET を起動し、UDF（ “potential_map.udf” ）を Open する。Open 時、画面には Record Numebr が **0** のデータが表示されているので Open 窓下のつまみを右にスライドさせ、右下の窓に Record Label である “BS_LAMELLA” が表示されるまで移動する。

3. 分子の作成（パイソンスクリプトの Load と編集ならびに実行）

ここでは、Atom1200 個からなるビーズスプリングモデルの分子 4 本からなる系を作成する手順について説明する。
なお、このシステムは **Action SILK** コマンド **SILK_CREATE_LinearPolymer_Bead_Spring_1_Home** を用いても作成することが出来る。コマンドの詳細は第 6 章参照。

(a) パイソンスクリプトの Load

GOURMET 上で “silk/bin/silk_use_lamella.py” を Load する。Python 窓に Load されたスクリプトが表示されていることを確認し、スクリプトの上部を参照する。

(b) Load されたパイソンスクリプトの編集

Load されたスクリプトの上部にユーザーが編集すべきセクション（**SECTION “USER DEFINITION”**）が記述されている。

- 出力先
  “USER DEFINITION” 内の **SUBSECTION “outputpath”** について説明する。

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

  self.engine.outDir には出力先のディレクトリを指定する。

  self.engine.cognacFileName には出力ファイル名（UDF 拡張子 : .udf を除く）を指定する。この例では “bs_lamella_in.udf” という **SILK** の出力、すなわち **COGNAC** 入力ファイルが作成される。

  self.engine.prjName には出力ファイルのプロジェクト名を指定する。

  self.engine.fileNumCom には出力ファイルを分割する（トポロジーを記述したファイルとその他の条件を記述したファイルに分割する）か否かのコマンドを記述する。
  self.engine.fileNumCom=”ONE_FILE”とした場合、単一のファイルにまとめられる。

- 分子の定義

“USER DEFINITION” 内の **SUBSECTION “system”** について説明する。
ここでは、**SILK** が用意しているテンプレート関数 **makeLinPolym(...)** を用いる。

```
##### 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)
#
```

各関数および値について次に説明する。

name（値:**”BS_lamella”**）··· 分子の名前。

numAtom（値:**1200**）··· 1 分子あたりの Atom の数。

numMol（値:**4**）··· 分子の本数。

atomName（値:**”Name1”**）··· Atom の名前。

atomTypeName（値:**”atom1”**）··· Atom type の名前。
UDF Path 名:**Molecular_Attributes.Atom_Type[0]** のデータ **Name** に対応している。

bondTypeName（値:**”bond1”**）··· Bond potential の名前。
UDF Path 名:**Molecular_Attributes.Bond_Potential[0]** のデータ **Name** に対応している。

angleTypeName（値:””）··· ビーズスプリングモデルを扱うので Angle は定義しない。よって何も入力しない。

torsionTypeName（値:””）··· ビーズスプリングモデルを扱うので Torsion は定義しない。よって何も入力しない。

interactionSiteTypeName（値:**”siteType1”**）··· Interaction site type の名前。
UDF Path 名: **Molecular_Attributes.Interaction_Site_Type[0]** のデータ **Name** に対応している。

(c) パイソンスクリプトの実行

編集したパイソンスクリプトの内容（出力先等）を確認し、スクリプトを実行する。実行結果についてはログウインドウに表示される。

4.7.2 入力データの編集と COGNAC の起動

1. **SILK** により作成した **COGNAC** 入力 UDF の読み込み

   GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから **SILK** において **self.engine.cognacFileName** で指定した、ファイル名 “bs_lamella_in.udf” を選択する。

2. ポテンシャルの確認

   起動画面の左側にはエクスプローラー様のデータ構造の各項目をクリックすることによってデータを参照（編集）する。

3. 初期構造の設定

   UDF Path 名 **Initial_Structure** で初期構造設定する。ここでは **Initial_Structure.Generate_Method.Method** に **Lamella** を指定して、**Initial_Structure.Generate_Method.Lamella** 以下のパラメータをセットする。パラメータの詳細に関しては §5.2.3 参照
   特に、**Initial_Structure.Initial_Unit_Cell.Cell_Size** を開き $\mathbf{a = b = 0.0, c = 20.0}$ になっていることを確認する。なお、Lamella generator においては結晶相／非晶相別々に密度を指定するので **Initial_Structure.Initial_Unit_Cell** の値は無視される。
   次に **Initial_Structure.Generate_Method.Lamella** を開き、**Lamella_Length** が **15.0** になっていることを確認する。これにより Z 軸方向のユニットセル長 **20.0** のうち **0.0-15.0** の範囲に結晶相が構築される。その他、**Initial_Structure.Generate_Method.Lamella.Random_Param**、**Initial_Structure.Generate_Method.Lamella.Helix_Param** の内容を確認する。特に先に述べたように各々の Density の設定を確認する。

4. 計算条件の設定

   UDF Path 名 **Simulation_Conditions** を編集することにより計算条件を設定する。設定したパラメータは以下のとおりである。

   (a) ダイナミクス条件

   UDF Path 名 · · · **Simulation_Conditions.Dynamics_Conditions** 以下のデータ値について説明する。

   Time.delta_T（設定値:**0.01**）· · · 1 ステップ当りの時間。

   Time.Total_Steps（設定値:**10000**）· · · シミュレーションの総ステップ数である。

   Time.Output_Interval_Steps（設定値:**1000**）· · · 出力のインターバルを示す。

   Temperature.Temperature（設定値:**0.1**）· · · 設定温度。

   Temperature.Interval_of_Scale_Temp（設定値:**100**）· · · 速度スケーリングのインターバルステップ数。

(b) ソルバー

UDF Path 名 · · · **Simulation_Conditions.Solver** 以下のデータ値について説明する。

Solver_Type（設定値:**Dyanimcs**）· · · ダイナミクス。
Dynamics.Dyanamics_Algorithm（設定値:**NVE**）· · · 実行するアルゴリズム。

*アルゴリズム **NVE** には、必要とする固有のパラメータは無い。固有のパラメータのないアルゴリズムとしてはその他 **SLLOD_T_Const** がある。

(c) 境界条件

Boundary_Conditions（設定値:**PERIODIC**（**a_axis**、**b_axis**、**c_axis** 全て））· · · 境界条件の設定。

(d) 計算するポテンシャル項の設定

Calc_Potential_Flags.Bond（設定値：**1**）· · · 結合伸縮ポテンシャルの計算条件。

Calc_Potential_Flags.Non_Bonding（設定値：**1**）· · · 分子間相互作用の計算条件。

(e) 出力データ項目の設定

UDF Path 名 · · · **Simulation_Conditions.Output_Flags** 以下のデータ値について説明する。

Structure.Position（設定値：**1**）
Structure.Velocity（設定値：**0**）
Structure.Force（設定値：**0**）· · · 各 Atom の座標、速度、力について出力の選択。座標のみ出力する。

5. UDF の Save

各パラメータについて編集操作を施した場合、編集結果が反映されるように **GOURMET** のメニューから Save をしなければならない。

6. **COGNAC** の起動

第 3 章　操作入門で示したような手順により **GOURMET Engine Run** コマンドより起動するか、あるいは、コマンドプロンプトあるいはシェルウインドウで、UDF を保存したディレクトリに移動し以下のコマンドを実行する。

```
cognac92 -I bs_lamella_in.udf -O bs_lamella_out.udf > bs_lamella.log
```

プログラムが終了すると、“bs_lamella_out.udf”（UDF 出力）、“bs_lamella_out.dat”（データ出力）、“bs_lamella.log”（ログファイル）が生成する

4.7.3　計算結果の表示と解析

分子構造の表示

計算結果を **GOURMET** に読み込んで、第 3 章　操作入門に示したような手順により、**GOURMET** 画面より、**Action command** を用いて、分子構造を表示することが出来る。
その際、color='mol' と指定すると、図 4.19 のように分子単位で色分けして表示される。

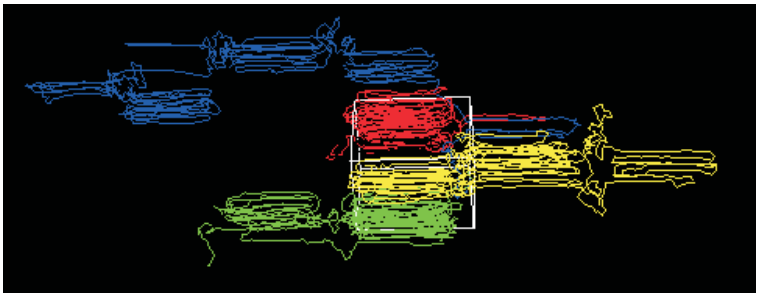


図 4.19: 分子単位で色分けして表示した例

Accelrys car フォーマットファイルへのエキスポート

ここでは、UDF に記述されている分子構造、座標データを Accelrys の分子構造を定義するフォーマットの一種、car file format で出力する方法を解説する。サンプルとして、スケールの整合の取りやすさから、上で作成したものと同様の方法で作成した CH_2 united atom のラメラ構造を例にとる。なお、同様のは **Action command : EXPORT_data...** を用いて行うことも出来る。詳細は 7.5 参照。

1. UDF の読み込み

   GOURMET Editor ウインドウより **File** → **Open...** を選択して、現れるファイルブラウザから、用意されている"out/ua_lamella_out.udf" を選択する。

2. 変換用スクリプトの読み込みと実行

   変換したい Record に移動した後、**Python:Load...** を選択し現れるファイルブラウザより "FileConvert.py" を選択して読み込み、**Run** を選択する。プログラムが終了すると読み込んだ UDF の存在するディレクトリに"ua_lamella_out.car" というファイルが生成される。
   このスクリプト内の scale parameter は長さ σ のスケーリングで、1σ に対応するÅ の値を設定する。

このようにして作成した car file は Accelrys 社 Material Studio(TM) 等の製品で読み込むことが可能である。ただし結合情報等を指定する mdf file は作成されないので、ソフトウエアによっては結合が正しく再現されない場合もある。
Accelrys 社のソフトウエアで読み込まれたデータは、編集により水素を付加し、ポテンシャルをアサインすれば Atomistic MD を行うことも可能である。

4.8 （事例 VII）DPD によるブロックコポリマーのミクロ相分離シミュレーション

ここでは、DPD シミュレーションによる AB ジブロックコポリマーの、ミクロ相分離シミュレーションの例を示す。

使用するファイルは特に指定するものを除いて、“sample/DPD” 以下にある。

4.8.1 Action SILK による入力データ作成

ここでは、**Action SILK** を用いて A5B5 ジブロックコポリマー 2,400 分子のトポロジー情報を作成する手順について説明する。

1. **SILK** 入力ファイル（“potential_map.udf”）

   まず最初に **SILK** 入力ファイル（“python/silk/sample/potential_map.udf”）を参照する。ここでは、10 番目のレコード（Record Number は **9**、Record Label は “DPD”）を使用する。

2. UDF（“potential_map.udf”）の Open

   GOURMET を起動し、**File** → **Open...** メニューから UDF（“potential_map.udf”）を Open する。画面には Record Number **0** のデータが表示されているので、**GOURMET** 窓下のつまみを右にスライドさせ、右下の窓に Record Label である “DPD” が表示されるまで動かす。

3. 分子の作成

   (a) **Action SILK** ポリマーテンプレートコマンドの選択
   読み込んでいる “potential_map.udf” の **Set_of_Molecules** のフォルダを右クリックして現れるコマンドより、**SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block...** を選択する。

   (b) **SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block...** の設定
   ここでは分子 A5B5 を 2,400 分子作成する。分子のアーキテクチャー以外に Atom のタイプ、bond potential のタイプ、interaction site のタイプ等もここで指定する。
   そのために、**SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block...** を選択して現れるウインドウで、作成するジブロックコポリマーのアーキテクチャおよびポテンシャルタイプ等を図 4.20 のように設定する。
   パラメータの詳細は、第 3 章および第 6 章参照。
   パラメータの入力が終わった後、**OK** をクリック。この時点ではまだ、**Set_of_Molecules** には書き込まれない。

4. **Set_of_Molecules** への書き込み

   このサンプルでは一種類のブロックコポリマーを作成するので、ここで定義した、分子を **Set_of_Molecules** へ書き込む。

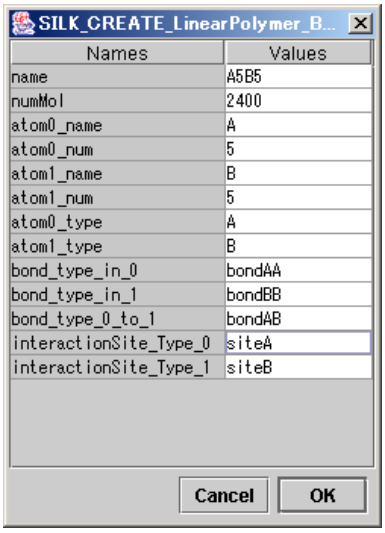


図 4.20: **SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block...** のパラメータ設定

再び、**Set_of_Molecules** のフォルダの右クリックにより現れるコマンドより、
SILK_OUT_SYSTEM_to_Set_of_Molecules... を選択し現れるウインドウで **Continue** を選択したまま **OK** をクリックすると “potential_map” の該当の Record に **Set_of_Molecules** が書き出される。ただし、この時点では **GOURMET** のメモリキャッシュ内に書き込まれるのみなので、Save しない限り”potential_map.udf” ファイルは更新されない。通常、雛型の”potential_map.udf” ファイル自体には **Set_of_Molecules** を追加しないほうがいい。

5. 新規の UDF ファイルの作成

“potential_map” に書き込まれた **Set_of_Molecules**、および雛型の計算条件を COGNAC の入力 UDF として用いるため、新規な UDF ファイルを作成して、そこに “potential_map” の該当の Record を書き出す。
そのために今度は **Editor** ウインドウに現れる “potential_map.udf” のルートのアイコンを右クリックして、現れるコマンド群より **SILK_UDF_CREATE...** を選択する。そこで、現れるウインドウで出力 UDF ファイル名を指定することにより、COGNAC 入力 UDF が作成される。ここではあとの説明のため、ファイル名を “A5B5_in.udf” としておく。

4.8.2 入力 UDF の編集と COGNAC の起動

次に、**Action SILK** で作成した **COGNAC** の入力 UDF である “A5B5_in.udf” を **GOURMET** で読み込み、計算条件等を編集した後、**COGNAC** を起動する。

1. 入力 UDF の読み込み

GOURMET Editor ウインドウより、**File** → **Open...** を選択して、現れるファイルブラウザから “A5B5_in.udf” を選択して読み込む。

2. 初期構造の設定

UDF Path 名 **Initial_Structure** で初期構造に関する情報を設定する。**Initial_Structure** 以下のデータの主なものを説明する。

Initial_Unit_Cell.Density･･･ 初期条件として設定される系の密度である。

ここでは無次元化された密度 **3.0** を入力している。

Generate_Method.Method･･･ 初期構造作成についての方法を記述。

Value の欄を左クリックすると、セレクトメニューが現れる。その中より **Random** を選択することにより、UDF Path 名 **Generate_Method.Random** 以下のデータが **COGNAC** によって参照される。

同じ階層に存在するその他のデータ（例えばUDF Path 名 **Initial_Structure.Generate_Method.Helix**）は無視されることに注意。

3. 計算条件の設定

UDF Path 名 **Simulation_Conditions** を編集することにより計算条件を設定する。設定したパラメータは以下のとおりである。

(a) ダイナミクス条件

UDF Path 名 **Simulation_Conditions.Dynamics_Conditions** 以下のデータ値について説明する。

Time.delta_T（設定値:**0.06**）･･･ 1 ステップ当りの時間を示す。

Time.Total_Steps（設定値:**10000**）･･･ シミュレーションの総ステップ数である。10,000 ステップでは相分離が平衡状態までに達しないので、100,000 ステップ程度に変更するとよいが、計算時間が長くなるので注意。

Time.Output_Interval_Steps（設定値:**1000**）･･･ 出力のインターバルを示す。

Temperature.Temperature（設定値:**1.0**）･･･ 設定温度。

Temperature.Interval_of_Scale_Temp（設定値:**0**）･･･ 速度スケーリングのインターバルステップ数。

***Temperature.Interval_of_Scale_Temp** を **0** にした場合、速度スケーリングは実行されない。DPD においては、遥動散逸定理により温度制御がなされるので速度スケーリングの必要は無い。

(b) ソルバー

UDF Path 名 **Simulation_Conditions.Solver** 以下のデータ値について説明する。

Solver_Type（設定値:**Dynamics**）··· ダイナミクス（**Dynamics**）、ミニマイズ（**Minimize**）のいずれかが選択出来る。ここでは **Dynamics** を選択。

Dynamics.Dynamics_Algorithm （設定値:**DPD**）··· 実行するアルゴリズム。

Dynamics.DPD.lambda（設定値:**0.65**）··· **DPD** の運動方程式を解く際のパラメータ。式 2.37 参照

(c) 境界条件
UDF Path 名 **Simulation_Conditions.Boundary_Conditions** 以下のデータ値について説明する。

Boundary_Conditions（設定値:**PERIODIC** （**a_axis**、**b_axis**、**c_axis** 全て））··· 境界条件の設定

(d) 計算するポテンシャル項の設定
UDF Path 名 **Simulation_Conditions.Calc_Potential_Flags** 以下のデータ値について説明する。

Calc_Potential_Flags.Bond（設定値：**1**）
Calc_Potential_Flags.Angle（設定値：**0**）
Calc_Potential_Flags.Torsion（設定値：**0**）··· 分子内相互作用の計算条件。DPD の場合、結合変角ポテンシャルの計算に関する項目:**Angle**、および結合二面角ポテンシャルの計算に関する項目:**Torsion** は、計算を行わない。したがって設定値を 0 にする。

Calc_Potential_Flags.Non_Bonding（設定値：**1**）··· 分子間相互作用の計算条件。

(e) 出力データ項目の設定
UDF Path 名 **Simulation_Conditions.Output_Flags** 以下のデータ値について説明する。

Statistics ··· 温度や圧力等の出力項目の選択。（選択された項目がデータおよびログファイルに出力される。UDF には選択の如何にかかわらずすべて出力される）

Structure.Position（設定値：**1**）
Structure.Velocity（設定値：**1**）
Structure.Force（設定値：**0**）··· 各 Atom の座標、速度、力について出力の選択。座標および速度を出力する。

4. UDF の Save
各パラメータについて編集操作を施した場合、編集結果が反映されるように **GOURMET** の **File** → **Save** メニューにより UDF data を Save しなければならない。

5. **COGNAC** の起動

第 3 章　操作入門で示したような手順により **GOURMET Engine Run** コマンドより起動するか、あるいは、コマンドプロンプトあるいはシェルウインドウで、UDF file のあるディレクトリに移動し、以下のコマンドを実行する。

```
cognac92 -I A5B5_in.udf -O A5B5_out.udf > A5B5.log
```

プログラムが終了すると、“A5B5_out.udf”（UDF 出力）、“A5B5_out.dat”（データ出力）、“A5B5.log”（ログファイル）が生成する起動時の引数等入力の詳細に関しては §5.1 を参照。

4.8.3　計算結果の表示と解析

分子構造の表示

計算結果を **GOURMET** に読み込んで、第 3 章　操作入門に示したような手順により、**GOURMET** 画面より、**Action command** を用いて、分子構造を表示することが出来る。
図 4.21 に **type** を **line**、**bc** を **atom** として表示した例を示す。

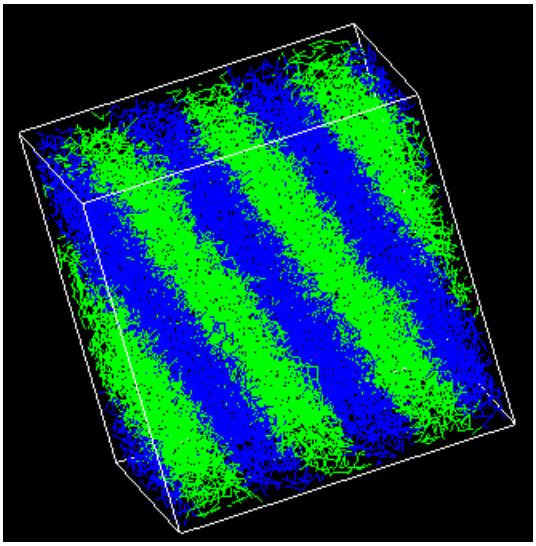


図 4.21: **type = 'line',bc = 'atom'** で表示した例

図に示されるように、規則正しいラメラ構造が得られることがわかる。

散乱関数の計算

Action メニュー **ANALYSIS_scattering_function...** を用いて **COGNAC** で計算した、原子の密度分布より散乱関数を計算することが出来る。

メニューを選択すると、図 4.22 のようなコマンドウインドウが現れるので、例えば図に示すようにパラメターをセットして、**Run** すると図 4.23 のような散乱関数の結果が表示される。パラメータの詳細に関しても”Readme” ファイルを参照。

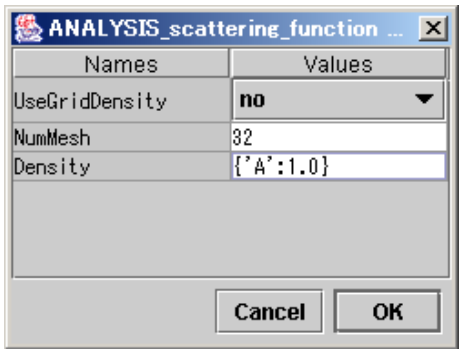


図 4.22: **ANALYSIS_scattering_function...** のパラメータ

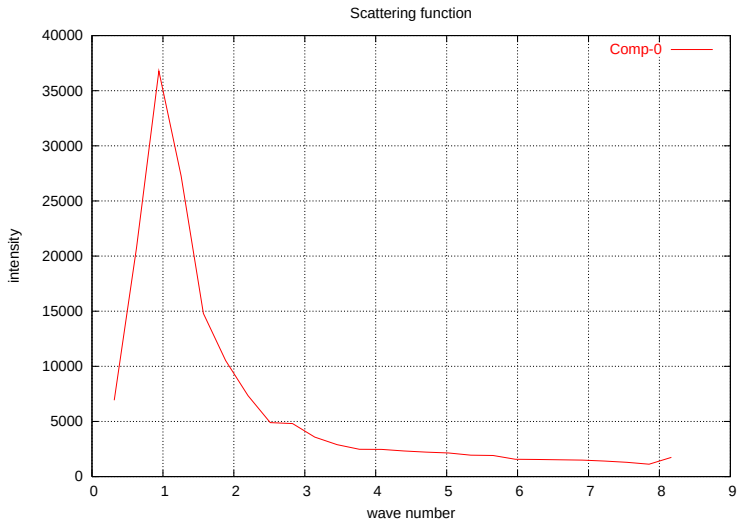


図 4.23: 散乱関数の表示

第5章　リファレンス

本章では、**COGNAC** の起動方法、ならびに **COGNAC** の持つ機能の詳細解説と入出力に用いる UDF の解説を行う。

5.1　起動方法

5.1.1　UDF 一覧

COGNAC が使用する UDF のリストをあげる。

- Definition UDF (“cognac92.udf”) ···　入出力 UDF の定義が記述されている。このファイル (“cognac92.udf”) は環境変数 **UDF_DEF_PATH** で定義されているディレクトリにおいてある必要がある。
  GOURMET からの起動、あるいは gourmetterm では自動的に設定されている。
- Input UDF ··· 入力データを持つ UDF
- Output UDF ··· 出力データが書かれる UDF
- Table UDF ··· Table Potenital を用いる際に必要な、データを持つ UDF
- Crystal UDF ··· 結晶構造を構築するため必要な、セルサイズ、対称操作、fractional coordinate 等のデータを持つ UDF

5.1.2　GOURMET からの起動

GOURMET Tool → **Engine Run...** メニューより開く **Engine Run** panel において、以下のパラメータを設定して Run ボタンにより起動する。

- **Run name**: ··· あらかじめ **COGNAC** を起動するために設定した名前を入力。設定の方法に関しては、**GOURMET** マニュアル参照
- **Server**: ··· **COGNAC** を起動するサーバーホスト名。ローカルで起動する場合は空欄でよい
- **Engine**: ··· **COGNAC** 実行ファイル (“cognac92”) を選択。**Run name** により正しく設定されていれば、変更の必要はない
- **Working Dir**: ··· エンジンを起動させる際のワーキングディレクトリ。入力 UDF のあるディレクトリを指定してはいけない
- **Parms**: ··· エンジン起動時に与えるパラメータ。**COGNAC** の場合通常は使用せず、空欄のままでよい
- **Input UDF**: ··· 入力 UDF ファイル名を指定。**Always Use Current UDF at Input UDF** をチェックしておけば **Engine Run** を起動した **Edit** window で読み込まれている UDF が自動的に指定される
- **Params UDF**: ··· **COGNAC** においては使用しない。空欄のままでよい

- **Restart UDF**: ··· エンジン起動時にリスタート UDF を指定する場合、ここで UDF ファイル名を指定する。**Set_of_Molecules** を入力 UDF と別ファイルから読み込む場合も、ここで指定する。
- **Output UDF**: ··· 出力 UDF ファイル名を指定。
- **Summary UDF**: ··· **Engine control panel** において計算経過表示を行うための、サマリデータ出力 UDF を指定する。任意のディレクトリ、ファイル名でよい
- **Logger**: ··· **COGNAC** においては使用しない。空欄のままでよい

5.1.3 コマンドラインからの起動と、起動時オプション

シェルプロンプト (unix/linux/cygwin) あるいはコマンドプロンプト (Windows) より、**COGNAC** のプログラム名を指定し、引数を与えて実行する。ただし、コマンドラインより実行する際には **PATH** その他の環境変数が設定されている必要がある。gourmetterm（"GOURMET_2007/bin" にあるバッチファイルあるいはシェルスクリプト）を用いると、これらの環境変数が自動的に設定されたシェル（コマンド）ウインドウが起動するので、これを利用するとよい。あるいは gourmetterm に書かれているように pfsetenv を起動すると環境設定が行われるので、起動シェルスクリプト等に加えておくとよい。

% 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 ··· 入力データ UDF
- output UDF (option) ··· 出力データ UDF
- restart UDF (option) ··· リスタート UDF
- number of thread (option) ··· **COGNAC** は Version8.3 より OpenMP による並列化が実装され、並列計算を行う際はスレッド数を指定する。デフォルト値は 1。
- order of precision of Structure output ··· トラジェクトリ出力において、座標、速度、力の出力の有効桁数を設定。デフォルトは 6。ただし、リスタート時の精度保障のため、最終レコードの桁数は 14 桁に固定されている。
- random seed ··· Random seed。指定しない場合はプログラムにより自動的に設定される

  （注意） Windows MinGW 環境にてコンパイルした実行ファイルを起動する場合、pthreadGC2.dll というファイルが必要である。このファイルはフリーでダウンロードできるので、COGNAC 実行ファイルと同じ、フォルダーにおいておくとよい。

（補足）

1. 出力 UDF を指定しない場合、出力は、**-R** によるリスタート UDF の指定があればリスタート UDF に、なければ**-I** で指定する入力 UDF に Append される
2. 出力 UDF に**-R** で指定するリスタート UDF と同じ UDF を指定した場合も、出力はリスタート UDF に Append される
3. **-R** でリスタート UDF を指定した場合、入力 UDF においてリスタート UDF から **Set_of_Molecules** を読み込む。同時に **Initial_Structure.Generate_Method.Method** が **RESTART** の場合を、座標データもここで指定する UDF から読み込まれる

5.1.4 その他の起動時オプション

COGNAC はエンジン実行以外に以下の引数をとる。

- -V ··· 実行モジュールのバージョンを表示する
- -C ··· 旧バージョンの Input UDF を最新バージョンに変換する
  （使用例）
  % cognac92 -I 旧バージョンの UDF ファイル名 -O 新たに作成される最新バージョンの UDF ファイル名 -C

5.1.5 終了方法

GOURMET Engine Run panel より **COGNAC** を起動した場合、**Engine Control panel** においてジョブの停止を行うことが出来る。

Engine Control panel において、**Stop** を選択した場合、計算中の最終ステップの結果を出力 UDF に書き出して、正常終了する。

Kill を選択した場合、プロセスがオペレーティングレベルで Kill されるので、出力 UDF は正常に書き込まれない場合がある。

5.2 入力 UDF 解説

入力 UDF は最大、以下の 8 つの構造体より構成されている

- **Simulation_Conditions**
  計算条件全般（温度、圧力、タイムステップ、アンサンブル等）の設定
- **Initial_Structure**
  初期構造の選択（random/helix/crystal/restart 等）
- **Molecular_Attributes**
  シミュレーションに用いる原子種、結合ポテンシャル等の設定
- **Interactions**
  非結合ポテンシャル、静電ポテンシャルおよび外場の設定
- **Set_of_Molecules**
  分子構造（結合情報、座標、ポテンシャルのアサイン等）の設定。別ファイルとして持つことも可能
- **Structure**
  Set_of_Molecules で定義した分子構造に対応した、Atom の情報（座標、速度、力）、およびユニットセルの情報を持つ。通常はリスタートデータとして別ファイルから読み込まれる。初期座標を新たに生成する場合は不要
- **React_Conditions**
  化学反応（結合の生成、解離）を取り扱うための設定。化学反応を考慮しないシミュレーションにおいては不要
- **Unit_Parameter**
  単位換算に関するスケーリングパラメータを持つ
- **Draw_Attributes**
  View 画面における描画に関するパラメータを持つ。**COGNAC** におけるシミュレーションの際には、使用されることはなく、入力 UDF で指定された値が、そのまま出力 UDF に出力される。

表 5.1〜5.8 に各パラメータのリストを記す。各パラメータの詳細については本文を参照のこと。

表 5.1: Simulation_Conditions

UDF パス名	意味
"Dynamics_Conditions"	基本的な計算条件の設定
"Dynamics_Conditions.Max_Force"	Atom に作用する力がこの値以上になると、シミュレーションを停止
"Dynamics_Conditions.Time"	トータルシミュレーションステップ、タイムステップ等
"Dynamics_Conditions.Temperature"	シミュレーション温度に関する設定（MD における温度制御アルゴリズム、あるいは温度スケーリングにおいて有効）
"Dynamics_Conditions.Pressure_Stress"	外部圧力、ストレスに関する設定
"Dynamics_Conditions.Deformation"	外場による変形、ずり流動に関する設定
"Dynamics_Conditions.Moment"	並進、回転モーメントの計算、出力、停止の設定
"Dynamics_Conditions.RATTLE"	Rattle(拘束) の設定
"Solver"	MD/MM の諸設定
"Solver.Solver_Type"	[Keyword] **DYNAMICS/MINIMIZE**
"Solver.Dynamics"	MD の諸設定
"Solver.Minimize"	MM の諸設定
"Boundary_Conditions"	境界条件の設定
"Boundary_Conditions.a_axis"	a 軸の境界条件
"Boundary_Conditions.b_axis"	b 軸の境界条件
"Boundary_Conditions.c_axis"	c 軸の境界条件
"Boundary_Conditions.Periodic_Bond"	周期境界条件を考慮した結合を含めるためのフラグ
"Calc_Potential_Flags"	ポテンシャルを計算するかどうかのフラグの設定
"Calc_Potential_Flags.Bond"	Bond Potential の計算を行うかどうかのフラグ
"Calc_Potential_Flags.Angle"	Angle Potential の計算を行うかどうかのフラグ
"Calc_Potential_Flags.Torsion"	Torsion Potential の計算を行うかどうかのフラグ
"Calc_Potential_Flags.Non_Bonding_Interchain"	分子間の Non bonding interaction の計算を行うかどうかのフラグ
"Calc_Potential_Flags.Non_Bonding_Intrachain"	分子内の Non bonding interaction の計算を行うかどうかのフラグ
"Calc_Potential_Flags.Non_Bonding_1_3"	1-3 pair の Non bonding interaction の計算を行うかどうかのフラグ
"Calc_Potential_Flags.Non_Bonding_1_4"	1-4 pair の Non bonding interaction の計算を行うかどうかのフラグ
"Calc_Potential_Flags.External"	External interaction の計算を行うかどうかのフラグ
"Calc_Potential_Flags.Electrostatic"	Electrostatic interaction の計算を行うかどうかのフラグ
"Calc_Potential_Flags.Tail_Correction"	Non bonding interaction の tail correction を行うかどうかのフラグ
"Output_Flags"	計算した諸量を UDF に出力するかどうかのフラグ
"Output_Flags.Statistics"	エネルギー、圧力等の物理諸量の出力
"Output_Flags.Structure"	座標、速度等のデータ
"Output_Flags.Averaged_Structure"	座標、速度等の区間平均データ
"Output_Flags.Correlation_Function"	自己相関関数の on the fly 出力
"Density_Output"	座標データより濃度分布を計算して出力する機能

表 5.1: Simulation_Conditions

UDF パス名	意味
”Density_Output.Calc_Grid_Density”	密度分布を計算するかどうかのフラグ
”Density_Output.Interval_of_Calc_Density”	密度を計算する座標データをサンプリングするインターバル
”Density_Output.Collect_Density_Steps”	密度を計算する座標データをサンプリングするステップ
”Density_Output.Num_of_Grid”	密度を計算する格子のサイズ (x,y,z)
”Density_Output.Radius”	密度分布を計算する際基準とする atom の有効半径
”Density_Output.Normalize_On”	規格化するかどうかのフラグ
”Constraint_Conditions”	Atom の拘束の設定
”Read_from_Restart”	拘束条件をリスタートファイルから読み込むかどうかのフラグ
”Constraint_Atom[]”	拘束する Atom のセット

表 5.2: Initial_Structure

UDF パス名	意味
”Initial_Unit_Cell”	ユニットセルの設定
”Initial_Unit_Cell.Density”	設定密度
”Initial_Unit_Cell.Cell_Size”	セルサイズ
”Initial_Unit_Cell.Shear_Strain”	Lees-Edwards 境界条件を用いる際の、初期の shear strain
”Read_Set_of_Molecules”	**Set_of_Molecules** の参照先の設定
”Read_Set_of_Molecules.UDF_Name”	**Set_of_Molecules** を読み込む UDF
”Read_Set_of_Molecules.Record”	**Set_of_Molecules** を読み込む Record No.
”Generate_Method”	初期構造の読み込み、あるいは発生方法の指定
”Generate_Method.Method”	[Keyword] 初期構造の選択
”Generate_Method.Restart”	リスタートモードに関する設定
”Generate_Method.Random”	非晶構造の生成に関する設定
”Generate_Method.Helix”	規則的ならせん構造の生成に関する設定
”Generate_Method.Lamella”	結晶ラメラ構造の生成に関する設定
”Generate_Method.Crystal”	（特定の UDF に記述された格子定数等に基づく）任意の結晶構造の作成に関する設定
”Relaxation”	初期構造の緩和の設定
”Relaxation.Relaxation”	初期構造の緩和の設定
”Relaxation.Method”	初期構造緩和のアルゴリズム [Keyword] **DYNAMICS/MINIMIZE**
”Relaxation.Max_Relax_Force”	緩和の際に atom にかかる力をカットする閾値
”Relaxation.Max_Relax_Steps”	緩和ステップの最大値
“Relaxation.Use_RATTLE”	緩和計算時に Bond RATTLE を用いるフラグ

表 5.3: Molecular_Attributes

UDF パス名	意味
"Atom_Type[]"	atom のタイプに関する設定
"Atom_Type[*].Name"	原子種の名前
"Atom_Type[*].Mass"	原子種の質量
"Bond_Potential[]"	結合長ポテンシャル（Bond Potential）に関する設定
"Bond_Potential[*].Name"	Bond Potential の名前
"Bond_Potential[*].Potential_Type"	Bond Potential のタイプ [Keyword]
"Bond_Potential[*].R0"	平衡結合長
"Bond_Potential[*].Harmonic"	Harmonic Bond Potential に関する設定
"Bond_Potential[*].FENE_LJ"	FENE+LJ Bond Potential に関する設定
"Bond_Potential[*].Gauss"	Gauss Bond Potential に関する設定
"Bond_Potential[*].Morse"	Morse Bond Potential に関する設定
"Bond_Potential[*].Bond_Polynomial"	多項式型の関数を持つ Bond Potential に関する設定
"Bond_Potential[*].DPD"	DPD の際用いられる Harmonic タイプの Bond Potential に関する設定
"Bond_Potential[*].Table_Bond"	Table 形式の Bond Potential に関する設定
"Bond_Potential[*].User_Bond"	ユーザー定義の Bond Potential に関する設定
"Angle_Potential[]"	結合角ポテンシャル（Angle Potential）に関する設定
"Angle_Potential[*].Name"	Angle Potential の名前
"Angle_Potential[*].Potential_Type"	Angle Potential のタイプ [Keyword]
"Angle_Potential[*].theta0"	平衡角
"Angle_Potential[*].Theta"	Theta Harmonic Angle Potential に関する設定
"Angle_Potential[*].Theta2"	Theta Harmonic 2 Angle Potential に関する設定
"Angle_Potential[*].Cosine"	Cosine Harmonic Angle Potential に関する設定
"Angle_Potential[*].Theta_Polynomial"	多項式型の関数を持つ Angle Potential に関する設定
"Angle_Potential[*].Table_Angle"	Table 形式の Angle Potential に関する設定
"Angle_Potential[*].User_Angle"	ユーザー定義の Angle Potential に関する設定
"Torsion_Potential[]"	二面角ポテンシャル（Torsion Potential）に関する設定
"Torsion_Potential[*].Name"	Torsion Potential の名前
"Torsion_Potential[*].Potential_Type"	Torsion Potential のタイプ [Keyword]
"Torsion_Potential[*].Cosine_Polynomial"	多項式型の関数を持つ Torsion Potential に関する設定
"Torsion_Potential[*].Amber"	Amber 型 Torsion Potential に関する設定
"Torsion_Potential[*].Dreiding"	Dreiding 型 Torsion Potential に関する設定
"Torsion_Potential[*].Table_Torsion"	Table 形式の Torsion Potential に関する設定
"Torsion_Potential[*].User_Torsion"	ユーザー定義の Torsion Potential に関する設定
"Interaction_Site_Type[]"	Interaction site のタイプ に関する設定
"Interaction_Site_Type[*].Name"	Interaction site type の名前
"Interaction_Site_Type[*].Num_of_Atoms"	Site を定義する Atom の数
"Interaction_Site_Type[*].Range"	list をつくる際、リストアップする最大半径
"Interaction_Site_Type[*].Rigid"	2 atoms 以上で Interaction site を定義する場合、そのサイトを剛体として自由度を落とすためのフラグ
"Electrostatic_Site_Type[]"	Electrostatic site のタイプ に関する設定
"Electrostatic_Site_Type[*].Name"	Electrostatic site type の名前
"Electrostatic_Site_Type[*].Type_Name"	Electrostatic site type のタイプ名
"Electrostatic_Site_Type[*].Polarizability"	Electrostatic site type の分極率

表 5.4: Interactions

UDF パス名	意味
"Pair_Interaction[]"	非結合相互作用（Pair Interaction）に関する設定
"Pair_Interaction[*].Name"	Pair Interaction の名前
"Pair_Interaction[*].Potential_Type"	Pair Interaction のタイプ [Keyword]
"Pair_Interaction[*].Site1_Name"	Pair Interaction を作用させる、対となる Site の名前
"Pair_Interaction[*].Site2_Name"	同上
"Pair_Interaction[*].Cutoff"	Cut off 距離
"Pair_Interaction[*].Scale_1_4_Pair"	1-4 atom 間の Pair Interaction のスケールファクター
"Pair_Interaction[*].Lennard_Jones"	Lennard Jones Pair Interaction に関する設定
"Pair_Interaction[*].Lennard_Jones_EV"	Lennard Jones with excluded volume Pair Interaction に関する設定
"Pair_Interaction[*].General_Lennard_Jones"	General Lennard Jones Pair Interaction に関する設定
"Pair_Interaction[*].Gay_Berne"	Gay Berne Pair Interaction に関する設定
"Pair_Interaction[*].GB_LJ"	Gay Berne - Lennard Jones Pair Interaction に関する設定
"Pair_Interaction[*].DPD"	DPD の際に用いられる Pair Interaction に関する設定
"Pair_Interaction[*].Morse"	Morse の際に用いられる Pair Interaction に関する設定
"Pair_Interaction[*].Bukingham"	Bukingham の際に用いられる Pair Interaction に関する設定
"Pair_Interaction[*].Table_Pair_Potential"	Table 形式の Pair Interaction に関する設定
"Pair_Interaction[*].User_Pair_Interaction"	ユーザー定義の Pair Interaction に関する設定
"External_Interaction[]"	外場による相互作用（External Interaction）に関する設定
"External_Interaction[*].Name"	External Interaction の名前
"External_Interaction[*].Potential_Type"	External Interaction のタイプ [Keyword]
"External_Interaction[*].Site_Name"	External Interaction を作用させる、対となる Site の名前
"External_Interaction[*].LJ_Wall"	Lennard Jones type flat wall に関する設定
"External_Interaction[*].LJ_Atomic_Wall"	Lennard Jones atomic type potential に関する設定
"External_Interaction[*].Static_Field"	Homogeneous field に関する設定
"External_Interaction[*].Density_Field"	Density field に関する設定
"External_Interaction[*].Velocity_Field"	Velocity field に関する設定
"External_Interaction[*].Total_Density_Constrain"	Total Density Constrain に関する設定
"External_Interaction[*].External_Angle"	External angle potential に関する設定
"External_Interaction[*].External_Torsion"	External torsion potential に関する設定
"External_Interaction[*].User_External_Field"	ユーザー定義の External Interaction に関する設定
"Electrostatic_Interaction[]"	静電相互作用に関する設定
"Electrostatic_Interaction[*].Name"	Electrostatic Interaction の名前
"Electrostatic_Interaction[*].Algorithm"	Electrostatic Interaction の計算アルゴリズム [Keyword]
"Electrostatic_Interaction[*].Dielectric_Constant"	システムの比誘電率
"Electrostatic_Interaction[*].Scale_1_4_Pair"	1-4 atom 間の Electrostatic Interaction のスケールファクター
"Electrostatic_Interaction[*].Cutoff_Coulomb"	Cutoff アルゴリズムに関する設定
"Electrostatic_Interaction[*].Reaction_Field"	Reaction Field に関する設定
"Electrostatic_Interaction[*].Ewald"	Ewald に関する設定
"Electrostatic_Interaction[*].Field_Electrostatic"	Field electrostatic アルゴリズムに関する設定

表 5.5: React_Conditions

UDF パス名	意味
”React_Flag”	化学反応を取り入れるかどうかのフラグ
”Atom_Exchange”	Atom Type の置換に関する設定
”Exchange_Type_Array[]”	Atom Type の置換タイプの設定
”Bond_Creation”	結合生成に関する設定
”Bond_Creation.Reactive_Atom[]”	反応に関わる Atom の設定
”Bond_Creation.Creation_Type_Array[]”	結合生成タイプの設定
”Bond_Creation.Potential_Assignment”	新規に生成する Angle/Torsion のポテンシャル設定
”Bond_Scission”	結合解離に関する設定
”Bond_Scission.Bond_Scission[]”	解離する bond の設定

表 5.6: Set_of_Molecules

UDF パス名	意味
”Molecule[]”	分子の配列
”Molecule[*].Mol_Name”	分子の名前
”Molecule[*].atom[]”	この分子（Molecule[*]）が持つ Atom の配列
”Molecule[*].bond[]”	この分子が持つ Bond の配列
”Molecule[*].angle[]”	この分子が持つ Angle の配列
”Molecule[*].torsion[]”	この分子が持つ Torsion の配列
”Molecule[*].interaction_Site[]”	この分子が持つ **interaction_Site** の配列
”Molecule[*].electrostatic_Site[]”	この分子が持つ **electrostatic_Site** の配列

表 5.7: Structure

UDF パス名	意味
”Position”	Atom の位置
”Position.mol[]”	**Set_of_Molecules** の **molecule[].atom[]** と同じ構造を持つ、Atom の位置を収めた構造体
”Velocity”	Atom の速度
”Velocity.mol[]”	**Set_of_Molecules** の **molecule[].atom[]** と同じ構造を持つ、Atom の速度を収めた構造体
”Force”	Atom にかかる力
”Force.mol[]”	**Set_of_Molecules** の **molecule[].atom[]** と同じ構造を持つ、Atom にかかる力を収めた構造体
”Unit_Cell”	Unit Cell の情報
”Unit_Cell.Density”	密度
”Unit_Cell.Cell_Size”	Unit Cell の情報
”Unit_Cell.Cell_Size.a”	a 軸長
”Unit_Cell.Cell_Size.b”	b 軸長
”Unit_Cell.Cell_Size.c”	c 軸長
”Unit_Cell.Cell_Size.alpha”	α 角
”Unit_Cell.Cell_Size.beta”	β 角
”Unit_Cell.Cell_Size.gamma”	γ 角
”Unit_Cell.Shear_Strain”	Lees-Edwards 境界条件の場合の shear strain 量

表 5.8: Unit_Parameter

UDF パス	意味
”Name”	ユニットセットの名前
”Comment”	コメント
”Mass”	Reduced mass
”Energy”	Reduced energy
”Length”	Reduced length

表 5.9: Draw_Attributes

UDF パス	意味
"Atom_Type[]"	属性を定義する **Atom_Type**
"Atom_Type.Name"	**Atom_Type** の名前
"Atom_Type.color"	色の指定
"Atom_Type.trancparency"	透明度の指定
"Atom_Type.radius"	半径の指定
"Bond_Potential[]"	属性を定義する **Bond_Potential**
"Bond_Potential.Name"	**Bond_Potential** の名前
"Bond_Potential.color"	色の指定
"Bond_Potential.trancparency"	透明度の指定
"Bond_Potential.radius"	半径の指定
"Molecule[]"	属性を定義する分子
"Molecule.Mol_Name"	分子の名前
"Molecule.color"	色の指定
"Molecule.trancparency"	透明度の指定
"Molecule.radius"	半径の指定

以下、構造体ごとに、入力データの説明を行う

5.2.1 注意事項

パラメータの型

個々の入力パラメータは float,int,string 等の決まった型のデータを取る。**GOURMET** で表示した場合、データとして与えるべき型がパラメータの名前とともに表示される。ただし **GOURMET** においては bool 型の変数はサポートしてない。**COGNAC** においては bool 型の変数の代用として short を用いており、short 型のパラメータには常に **1**(true), **0**(false) の値を入力するようにする。

入力文字列の扱い

例えば **Dynamics** の際に選択するアルゴリズムなど、予約語として登録されている keyword を入力する場合は大文字、小文字の区別は必要ない。また文字列の前後あるいは文字列中に空白文字が有る場合は、すべて取り除かれて解釈される。

一方 **Atom_Name** の様に任意の string を入力できる項目に関しては大文字、小文字の区別をする。入力 UDF において、リスタート UDF 等を指定する際の UDF 名も大文字、小文字は区別される。

5.2.2 Simulation_Conditions

Dynamics_Conditions

基本的な計算条件の設定

- **Time** : トータルシミュレーションステップ、タイムステップ等の設定

  - **delta_T** ··· タイムステップ Δt
  - **Total_Steps** ··· トータルステップ
  - **Output_Interval_Steps** ··· データ出力を行うインターバル

- **Temperature**：シミュレーション温度に関する設定（MD における温度制御アルゴリズム、あるいは温度スケーリングにおいて有効）
  - **Temperature** ··· 設定温度
  - **Interval_of_Scale_Temp** ··· 温度スケールを行うインターバル

- **Pressure_Stress**：外部圧力、ストレスに関する設定
  - **Pressure** ··· 設定圧力
  - **Stress** ··· 設定ストレステンソル (xx,yy,zz,yz,zx,xy)

- **Deformation**：外場による変形、ずり流動に関する設定
  - **Method** ··· 変形あるいはずり流動の方法の指定。[Keyword] **Lees_Edwards**/**Cell_Deformation**
  - **Lees_Edwards**：Lees-Edwards アルゴリズムによるずり流動の設定
    - **Method** ··· [Keyword] **Steady** (定常ずり。旧バージョンの **Static** より変更))/**Dynamic**(動的ずり)
    - **Steady.Shear_Rate** ··· 定常ずりの際のずり速度。変化率 $d\gamma_{yx}/dt[\tau^{-1}]$ であたえる。
    - **Dynamic.Amplitude** ··· 動的ずりの際の最大変形率（最大ひずみ）
    - **Dynamic.Frequency** ··· 動的ずりの際の周波数 $[\tau^{-1}]$
  - **Cell_Deformation**：セル変形の設定
    - **Method** ··· セル変形の方法の指定。[Keyword] **Deformation_Rate**/ **Simple_Elongation** /**Oscillation**
    - **Interval_of_Deform** ··· 実際にセル変形を行うインターバル
      delta_T を 1 ステップとする。毎ステップ変形させるのが望ましいが（**Interval_of_Deform=1**）、変形操作のための計算時間がかかるので、シミュレーションに支障がない程度に間隔をあけてもよい。このステップ間隔をどの様に設定しても、**Deformation_Rate** あるいは **Simple_Elongation** で与える速度は変化しない
    - **Deform_Atom** ··· セルを変形する際に、個々の原子も同時にアフィン変形させるかどうかのフラグ
    - **Deformation_Rate** ··· セルの変形速度テンソル。変化率 $d\epsilon/dt[\tau^{-1}]$ で定義される
    - **Simple_Elongation.Elongation_Rate** ··· z 軸方向への伸長変形の速度。$dL_z/dt[\sigma\tau^{-1}]$ で定義される。
    - **Simple_Elongation.Poisson_Ratio** ··· 伸長時のボアソン比 (**0-0.5**)。入力値に応じて x,y 方向のセルサイズが変形する
    - **Simple_Elongation.Axis** ··· 伸長モード。[Keyword] **z**（z 軸方向の一軸伸長）/ **xy**（xy 平面の等二軸伸長）

    - **Oscillation.Amplitude** ··· 動的伸張の際の最大変形率（最大ひずみ）
    - **Oscillation.Frequency** ··· 動的伸張の際の周波数 $[\tau^{-1}]$

    - **Oscillation.Poisson_Ratio** ··· 動的伸張の際のボアソン比 (**0-0.5**)。入力値に応じて x,y 方向のセルサイズが変形する

  - **Moment**： 並進、回転モーメントの計算、出力、停止の設定。ただしモーメントの停止の機能は Constraint Atom がある場合、動作は保証されない
    - **Interval_of_Calc_Moment** ··· モーメントを計算、出力、停止する頻度。**Calc_Moment = true(1)** の時のみ有効
    - **Calc_Moment** ··· モーメントを計算するかどうかのフラグ
    - **Print_Moment** ··· モーメントをログに出力するかどうかのフラグ。**Calc_Moment = true(1)** の時のみ有効
    - **Stop_Translation** ··· 並進モーメントを停止するかどうかのフラグ。**Calc_Moment = true(1)** の時のみ有効
    - **Stop_Rotation** ··· 回転モーメントを停止するかどうかのフラグ。
      Calc_Moment = true(1) の時のみ有効

      （注意）**COGNAC** において、回転モーメントは、周期境界条件を作用させて、個々の分子の重心がセル内に存在するイメージの座標を元に計算している。実座標から得られる回転モーメントとは異なるので注意。

  - **RATTLE** : RATTLE の設定
    - **Bond** ··· Bond RATTLE を適用するかどうかのフラグ
    - **Bond_Potential_Name** ··· RATTLE を適用する **Bond_Potential_Name** のリスト。指定の無い場合はすべての Bond に RATTLE を適用する。
    - **Angle** ··· Angle RATTLE を適用するかどうかのフラグ。**Bond=true(1)** である時のみ有効
    - **Angle_Potential_Name** ··· RATTLE を適用する **Angle_Potential_Name** のリスト。指定の無い場合はすべての Angle に RATTLE を適用する。
    - **Threshold** ··· 収束判定値
    - **Use_RATTLE** ··· 緩和計算時に Bond RATTLE を用いるフラグ。カテネーション時に結合長が不自然に伸びてしまう現象を回避する

- **Max_Force** ··· MD シミュレーション中に atom に作用する force がこの値以上になると、シミュレーションを停止する。発散して無意味なシミュレーションを行わないために設定する

Solver

MD/MM の諸設定

- **Solver_Type** ··· [Keyword] **DYNAMICS/MINIMIZE**
- **Dynamics**： MD の諸設定
  - **Dynamics_Algorithm** ··· [Keyword]

```
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
```

各 keyword の意味に関して説明する。詳しくは §2.3 を参照のこと。

* **NVE**
  マイクロカノニカルアンサンブル。温度スケールを行うことは可能また、NVE に加えて Lees-Edwards 境界条件 [21] を用いてせん断流動を与えることも可能。
* **NVT_Nose_Hoover**
  Nose-Hoover 法による温度制御
* **NVT_Berendsen**
  Loose coupling 法による温度制御
* **NVT_Kremer_Grest**
  Random force(Langevin dynamics) による温度制御
* **NPH_Andersen**
  Andersen 拡張ハミルトニアン法による圧力制御
* **NPH_Parrinello_Rahman**
  Parrinello-Rahman 拡張ハミルトニアン法による異方的圧力制御。ユニットセル角度を固定してユニットセル長のみを独立に可変可
* **NPH_Brown_Clarke**
  Loose coupling 法による異方的圧力制御。ユニットセル角度を固定してユニットセル長のみを独立に可変可
* **NPT_Andersen_Nose_Hoover**
  Andersen 拡張ハミルトニアン法による圧力制御　+ Nose-Hoover 法による温度制御。ユニットセル等方変形のみ
* **NPT_Andersen_Kremer_Grest**
  Andersen 拡張ハミルトニアン法による圧力制御　+ Random force による温度制御。ユニットセル等方変形のみ。
* **NPT_Parrinello_Rahman_Nose_Hoover**
  Parrinello-Rahman 拡張ハミルトニアン法による圧力制御 + Nose-Hoover 法による温度制御。ユニットセル異方変形可。ユニットセル角度を固定してユニットセル長のみを独立に可変可

* **NPT_Parrinello_Rahman_Kremer_Grest**
  Parrinello-Rahman 拡張ハミルトニアン法による圧力制御 + Random force による温度制御。ユニットセル角度の可変不可でユニットセル長は独立に可変可。
* **NPT_Berendsen**
  Loose coupling 法による圧力、温度制御。ユニットセル等方変形のみ。温度制御は NVT_Berendsen アルゴリズムに従う。圧力制御は 2.20 式で得られる μ により、原子の座標とユニットセルサイズをスケールする。
* **NPT_Brown_Clarke**
  Loose coupling 法による圧力、温度制御。ユニットセル異方変形可。ユニットセル角度を固定してユニットセル長のみを独立に変形することも可能。温度制御は **NVT_Berendsen** アルゴリズムに従う。圧力制御は Berendsen のアルゴリズムを拡張し、2.21 式に従う。
* **SLLOD_T_Const**
  SLLOD + Lees-Edwards 境界条件でせん断流動を与え、Nose-Hoover 法により温度制御 **COGNAC** においては SLLOD によりせん断流動を加える場合は、常に温度制御を行う。Version 8.0 より温度制御のアルゴリズムを拘束法より Nose-Hoover 法に変更した。
* **SLLOD_PT_Const**
  SLLOD + Lees-Edwards 境界条件でせん断流動を与え、Nose-Hoover 法により温度制御。同時に Parrinello-Rahman 法により法線方向の応力を制御。すなわち $\dot{\gamma}_{yx}$ の場合、z 方向の応力を制御するためユニットセル c 軸のみ変化する。Version 8.0 より圧力制御のアルゴリズムを Loose-Coupling 法より Parrinello-Rahman 法に変更した。
* **DPD**
  Dissipative particle dynamics。§2.4 参照

以下に Dynamics で指定するパラメータをあげる。実際の入力では選択した **Dynamics_Type** の各オプション名のフォルダー内のパラメータを設定するが、同意味のパラメータが有るのでここでは整理して説明する

* **Q** ··· Nose Hoover 法による温度制御の際に用いる、熱浴とのカップリングコンスタント
* **tau_T** ··· Loose coupling 法による温度制御の際に用いる、カップリングコンスタント
* **Friction** ··· Kremer-Grest 法による温度制御の際に用いる、Friction constant
* **Cell_Mass** ··· 拡張ハミルトニアン法による圧力制御の際に用いる、ユニットセルの仮想質量。質量 M の次元を持つ絶対量を取る
* **Fix_Angle** ··· 非等方圧力制御の際に用いる、ユニットセル角度を固定するかどうかのフラグ。true(1) で固定。
* **Fix_Cell** ··· 非等方圧力制御の際に用いる、固定するセル辺の指定。x（a 軸のみ固定）, yz（bc 軸を固定）等軸, x_iso（a 軸を固定すると同時に bc 軸の比率を固定して変形）, iso_xy（ab 軸の比率を固定して変形）などを入力。Fix_Angle = true(1) の時のみ有効
* **tau_P** ··· Loose coupling 法による圧力制御の際に用いる、カップリングコンスタント。ただし Berendsen アルゴリズムにおいては等温圧縮率 $\beta[\mathrm{P}(=\mathrm{ML}^{-1}\mathrm{T}^{-2})^{-1}]$ を含んだ $\tau_p/\beta[\mathrm{ML}^{-1}\mathrm{T}^{-1}]$ の次元を取る（[16] 参照）。また Brown-Clarke アルゴリズムにおいては 2.21 式の m に相当し、$[\mathrm{ML}^{-2}\mathrm{T}^{-1}]$ の次元を取る（[20] 参照）
* **lambda** ··· DPD の際の、時間積分に用いられる修正 velocity Verlet の係数。2.37 式参照

- **Minimize**：MM の諸設定

– **Minimize_Algorithm** ··· [Keyword]

`Steepest_Descent ...` 最急降下法
`Conjugate_Gradient ...` 共役勾配法
`Cascade ... Steepest_Descent` と `Conjugate_Gradient` を連続して行う

– **Max_Iteration** ··· 最大の iteration 回数
– **Converge_Force** ··· 収束判定値。すべての原子にかかる力がこの値以下になると収束
– **Output_Interval_Steps** ··· データ出力のインターバル
– **Cascade.SD_Max_Iteraction** ··· Cascade アルゴリズムの際の Steepest Decsent の最大 iteration
– **Cascade.SD_Converge_Force** ··· Cascade アルゴリズムの際の Steepest Decsent の収束判定値
– **Strain_Tensor[]** ··· 初期ユニットセルに対して与える、変形テンソル。変形 →minimize を配列要素の数繰り返す

Boundary_Conditions

境界条件の設定

- **a_axis,b_axis,c_axis** ··· 各々のセル軸の境界条件

  [Keyword] **NONE/PERIODIC/REFLECTIVE1/REFLECTIVE2**

  – **NONE**
  境界条件無し
  – **PERIODIC**
  周期境界条件。**COGNAC** では 2 次元および 3 次元周期境界条件（立方体セル、直方体セル、斜方セル）をサポート
  – **REFLECTIVE1/REFLECTIVE2**
  土井プロジェクトにおいて新規に開発された、スタガード反射境界条件 [44] をサポート。
  REFLECTIVE1 は $r = \mathrm{Cell}_{max}$ の面のみスタガード反射境界条件を考慮し、$r = 0$ の面は境界条件を考慮しない。
  一方 **REFLECTIVE2** は $r = 0, r = \mathrm{Cell}_{max}$ の両面にスタガード反射境界条件を考慮する

  (注意)

  – **PERIODIC** の指定の場合で、初期の shear strain あるいは shear flow が設定された場合、自動的に Lees-Edwards 境界条件が適用される
  – **COGNAC** の実装上の仕様により、**NONE** を指定した場合に、ユニットセルの外部に atom を配置した場合の動作は保証されない。

- **Periodic_Bond** ··· 周期境界条件考慮した結合をとりこむためのフラグ。**true(1)** の時、定義されている結合長、結合角、二面角はすべて周期境界条件を考慮し、最短の原子間距離に基づいて計算される。

Calc_Potential_Flags

ポテンシャルを計算するかどうかのフラグの設定

- **Bond, Angle, Torsion, Non_Bonding_Inter, Non_Bonding_Intra, Non_Bonding_1_3, Non_Bonding_1_4, External, Electrostatic**

  true(1) の時、各々のポテンシャル項を計算する。ただし **Non_Bonding_1_3**（1-3 pair の非結合相互作用）は、**Angle** および **Non_Bonding_Inter** あるいは **Non_Bonding_Intra** が **true** の時にのみ有効。すなわち **Angle=false** の時は **1_3_Non_Bonding** は常に計算される。また、**Non_Bonding_Inter** および **Non_Bonding_Intra** の **false** の時は **1_3_Non_Bonding** も計算されない。
  静電力の計算を指定した場合も、同様の基準で 1-3 electrostatic potential の計算がオン、オフされる。**Non_Bonding_1_4** に関しても、同様に **Torsion** の設定に依存する。

- **Tail_Correction**
  Non bonding interaction の tail correction を行うかどうかのフラグ。Lennard-Jones および General Lennard-Jones タイプの Non bonding interaction にのみ有効。

Output_Flags

計算した諸量を UDF に出力するかどうかのフラグ

- **Statistics** ··· エネルギー、圧力等の物理諸量の出力
  true(1) の時 **Energy, Temperature, Pressure, Stress, Volume, Density, Cell, Wall_Press, Energy_Flow** 各々の計算結果が出力される。

  （**注意**）この出力フラグは、標準出力、データファイル出力にのみ有効で、UDF 出力には上記の項目がすべて出力される。

- **Structure** ··· 座標、速度等のデータ
  各々のフラグが **true(1)** の時 **Position, Velocity, Force** の値が出力される。

- **Averaged_Structure** ··· 座標、速度等の区間平均データ
  各々のフラグが **true(1)** の時 **Position, Velocity, Force** の **Dynamics_Conditions.Time.Output_Interval_Steps** で指定された区間の平均値が出力される。

- **Correlation_Function** ··· 自己相関関数の on the fly 計算を行い出力をするためのフラグ。現バージョンでは **Stress** の自己相関関数のみをサポート。出力する際は **true(1)** に指定。

Density_Output

座標データより濃度分布を計算して出力する機能。濃度分布計算の方法に関しては §2.10 参照。

- **Calc_Grid_Density** ··· 密度分布を計算するかどうかのフラグ

- **Interval_of_Calc_Density** ··· 密度を計算する座標データをサンプリングするインターバル

- **Collect_Density_Steps** ··· 密度を計算する座標データをサンプリングするステップ
  Output_Interval で指定されるステップの直前から **Collect_Density_Steps** だけ遡る区間において、**Interval_of_Calc_Density** で指定されたインターバルでサンプリングを行う。

- **Radius** ··· 密度分布を計算する際に基準とする atom の有効半径

- **Normalize_On** ··· 規格化するかどうかのフラグ
- **Num_of_Grid** ··· 密度を計算する格子のサイズ (x,y,z)

Constraint_Conditions

Atom の拘束条件の設定

- **Read_from_Restart** ··· 拘束条件をリスタートファイルから読み込むかどうかのフラグ
  Initial_Structure において、**Set_of_Molecules** の読み込みとして指定した、あるいは起動時に-R で指定した udf より **Constraint_Atom**[] を読み込む
- **Constraint_Atom**[] ··· 拘束する Atom のセット。配列データとして持つ
  - **Index** ··· 拘束する Atom の指定
    - **Mol_Intex** ··· molecule の Index
    - **Atom_Intex** ··· atom の Index
  - **Constraint_Axis** ··· 拘束する軸
    - **x,y,z** ··· **YES** の場合、指定する軸方向のダイナミクスが拘束される。**NO** と指定された軸は拘束を受けない。
  - **Method** ··· 静的 **Steady** ／動的 **Dynamic** 拘束の選択 [Keyword]
    動的拘束において Atom は初期座標を原点として、振幅運動をする
  - **Steady.Velocity** ··· 拘束する速度。常に指定された速度で移動する。(0.0,0.0,0.0) の場合、Atom は固定される
  - **Dynamic.Amplitude** ··· 動的拘束の際の振幅
  - **Dynamic.Frequency** ··· 動的拘束の際の周波数

5.2.3 Initial_Structure

初期構造の設定

Initial_Unit_Cell

ユニットセルの設定

- **Density** ··· 設定密度
- **Cell_Size** ··· セルサイズ立方体、直方体セルの場合、ユニットセルの a,b,c は各々x,y,z 軸に対応する。

（注意） **Cell_Size** がセットされていると **Density** を無視して **Cell_Size** の入力に従い、ユニットセルを作成する。ただし、以下の例外がある。

- **alpha,beta,gamma** のいずれかの値がゼロの場合 ··· 直方体セルが作成される

  - **a,b,c** のいずれか 1 辺の値がゼロの場合 ··· 密度が **Density** で指定した値になるように、ゼロにセットされている辺の長さを計算して設定。ただし、**alpha = beta = gamma = 90.0deg.** の時のみ有効
  - **a,b,c** のいずれか 2 辺の値がゼロの場合 ··· 密度が **Density** で指定した値になるように、ゼロにセットされている辺の長さを計算して設定。2 辺の長さは等しくする。ただし **alpha = beta = gamma = 90.0deg.** の時のみ有効

- **Shear_Strain** ··· Lees-Edwards 境界条件を用いる際の、初期の shear strain を設定する

Read_Set_of_Molecules

- **UDF_Name** ··· **Set_of_Molecules** を読み込む UDF。空欄の場合入力 UDF より読み込む
- **Record** ··· **Set_of_Molecules** を読み込む Record No.
  反応を考慮したシミュレーションの結果を読み込む場合を除いて、通常 **Set_of_Molecules** は Initial data 部 (Record No.=**-1** で指定) にしか出力されていない。反応を伴うシミュレーションの最終出力よりリスタートする場合、最終レコードの値以上の値を指定する最終レコードの **Set_of_Molecules** を読み込む。

Generate_Method

初期構造を読み込み、あるいは発生させる方法を指定する

- **Method** ··· 初期構造の選択。[Keyword]

```
Restart ... 指定した UDF から座標、速度を読み込む
Random ... 非晶構造を生成する。多相構造を作成する場合は、Random を選択し、
           Random のオプションで諸条件を指定する。
Helix ... 規則的に配置されたらせん構造を生成する
Lamella ... 半結晶ラメラ構造を生成する
Crystal ... 格子定数、対称操作、fractional coordinate 等を UDF から読み込み、
            任意の結晶構造を作成する
```

- **Restart**
  - **UDF_Name** ··· リスタート UDF 名。空欄の場合 **Read_Set_of_Molecules** で指定された UDF よりデータを読み込む
  - **Record** ··· リスタートするレコード No.。リスタート UDF から指定するレコードのデータを読み込む。負の値あるいは存在するレコード数より大きな値を指定した場合、リスタート UDF の最後のレコードを読み込む
  - **Restore_Cell** ··· セルサイズをリスタート UDF から読み込むかどうかのフラグ
  - **Restore_Velocity** ··· atom の速度をリスタート UDF から読み込むかどうかのフラグ

- **Random**
  - **Fix_Angle** ··· 結合角を対応する angle potential で与えられた平衡角に固定するかどうかのフラグ

– **Num_Torsion_State** ··· 二面角を指定するオプション。**Fix_Angle=true(1)** の時に有効

```
Num_Torsion_State = 0 ... ランダムに発生
Num_Torsion_State = 1 ... トランスのみ
Num_Torsion_State = 2 ... トランスーシス
Num_Torsion_State = 3 ... トランスーゴーシュ
```

– **Delta_E** ··· **Num_Torsion_State = 2** or **3** の場合のトランスーシス、あるいはトランスーゴーシュのエネルギー差
トランス=0 とした場合の、シスあるいはゴーシュのエネルギーの値を指定する

– **Build_Temp** ··· **Num_Torsion_State = 2** or **3** の場合、2 面角を発生させる温度
Delta_E と **Build_Temp** を用いてボルツマン因子より確率を計算

– **Density_Bias**[]：Atom_Type による濃度場拘束モンテカルロ法による初期構造の生成。拘束を与える Atom type の数、配列データとして持つ

  * **Atom_Name** ··· 拘束を与える Atom type name
  * **UDF_Name** ··· 濃度場を持つ **SUSHI**、**Muffin_phaseseparation** あるいは **Gird_Density** データを持つ **COGNAC** の UDF ファイル名
  * **Component_Index** ··· 拘束を与えるベースとなる **SUSHI** 出力 UDF 内の **SUSHIOutput.phi.value**[].**comp**[] の Index。**Muffin_phaseseparation** の場合は **Name** に **VolumeFraction** が指定されている field の **field.scalar_field**[].**value**[].**comp**[] の Index。**COGNAC** の場合は、**Grid_Density.phi.value**[].**comp**[] の Index。
  * **Max_Retry** ··· モンテカルロの最大試行回数

– **Node_Density_Bias**[]：個々の Atom の濃度場を用いた濃度場拘束モンテカルロ法による初期構造の生成。拘束を与える Molecule の数を要素数とする、配列データとして持つ

  * **Molecular_Name** ··· 拘束を与える Molecule name
  * **UDF_Name** ··· セグメント濃度場を持つ **SUSHI** UDF ファイル名
  * **Start_Index** ··· Atom Index = **0** に対応する **SUSHI** 出力 UDF 内の **segment_volume_fraction.value**[].**comp**[] の Index
  * **End_Index** ··· Atom Index = end に対応する **SUSHI** 出力 UDF 内の **segment_volume_fraction.value**[].**comp**[] の Index
  * **Scale_Param** ··· MD atom/SCF segment の比
  例えばビーズースプリングモデルで 1 atom of MD = 1 segment of SCF とする場合は 1.0。より微視的な MD モデルを用いた場合は 1 以上の値を与える。
  * **Max_Retry** ··· モンテカルロの最大試行回数

– **Meso_Ratio** ··· **Main_Chain** 中に Chiral な Atom がある場合、meso diad の確率。
Chirality が指定されていると無視される

– **Fix_End_Position**[]：分子の先頭の原子 (Atom Index = **0**) の座標を指定する

  * **Mol_Index** ··· 先頭の原子座標を指定する分子の Index
  * **Position** ··· 指定する座標値 (x,y,z)

- **Helix**
  - **Lattice_Type**：格子情報の設定
    - **Type** ··· 単純立方格子、体心立方格子等の設定 [Keyword]

      `SC ...` 単純立方格子
      `FCC ...` 面心立方格子
      `BCC ...` 体心立方格子
      `Manual ...` 体心立方格子様であるが、中心となる座標を任意に指定できる

    - **Manual.X_Shift** ··· **Manual** の場合、y 軸方向の奇数列に配置する原子の x 軸方向のシフト量。x 軸方向の格子間隔を 1 とする
    - **Manual.Z_Shift** ··· **Manual** の場合、y 軸方向の奇数列に配置する原子の z 軸方向のシフト量。z 軸方向の格子間隔を 1 とする
      X_shift = Z_shift = 1/2 の場合 BCC と同じになる
  - **Num_X_Cell** ··· 単位格子を x(a) 軸方向に並べる数
  - **Num_Y_Cell** ··· 単位格子を y(b) 軸方向に並べる数

    （注意）

    1. Type が **Manual** の場合は x,y 方向に並べる分子の数の指定を意味し、必ずしも単位格子とは一致しない
    2. ユニットセルのサイズと **Num_X/Y_Cell** により分子間隔が計算される。よって例えば **Type=”SC”** と指定しても厳密には立方格子にならない場合も作成可能
    3. z 軸方向の単位格子数は **Num_X/Y_Cell** とトータル分子数により自動的に計算される

  - **Fix_Angle** ··· 結合角を、対応する angle potential で与えられた平衡角に固定するかどうかのフラグ
  - **Torsion[]** ··· らせん構造を構築する場合の二面角。トランス=0。複数の二面角の連続によるらせんの場合、二面角が出現する順に値を配列データとして持つ
  - **Rotate1** ··· y 軸方向の偶数列の、z 軸を回転軸とした分子の回転角
  - **Rotate2** ··· y 軸方向の奇数列の、z 軸を回転軸とした分子の回転角の末端を構成する Atom Index の対

  - **Orient_Vector[]** ··· z 軸方向に配向させるベクトル
    例えば C0-C1-C2-C3-C4-... と定義されるポリエチレンユナイティッドアトムの場合 **Orient_Vector[0]**、**Orient_Vector[1]** を各々 **0,2** と指定してオールトランスのヘリックスをモデリングすると、C0 - C2 - C4 -が z 軸方向に配向したヘリックスが生成する。ちなみにこの場合 **0,4** と指定しても結果は同じ。この指定のない場合は Index=0 と Index=最大値の 2 原子間を結ぶベクトルを z 軸に配向させるベクトルとする。
  - **Meso_Ratio** ··· **Main_Chain** 中に Chiral な Atom がある場合、meso diad の確率。
    Chirality が指定されていると無視される
  - **Inverse_Molecules** ··· 分子を反転させて配置する際のオプション
    - **NONE** ··· 反転はなし。すべて同一の方向に配置。

  - **Neighbour_by_Neighbour** ··· 隣り合う分子で配置を反転する
  - **Line_by_Line** ··· b 軸方向の列単位で分子を反転する
  - **Layer_by_Layer** ··· c 軸方向の層単位で分子を反転する

- **Lamella**
  - **Lamella_Length** ··· ラメラ結晶相の長さ (z 軸方向)
  - **Fix_Angle** ··· 結合角を、対応する angle potential で与えられた平衡角に固定するかどうかのフラグ
  - **Meso_Ratio** ··· Main_Chain 中に Chiral な Atom がある場合、meso diad の確率。Chirality が指定されていると無視される
  - **Random_Param.Density** ··· 非晶部の密度
  - **Random_Param.Num_Torsion_State** 二面角を指定するオプション。**Fix_Angle=true(1)** の時に有効
  - **Random_Param.Delta_E** ··· **Num_Torsion_State = 2** or **3** の場合のトランスーシス、あるいはトランスーゴーシュのエネルギー差
    トランス=0 とした場合の、シスあるいはゴーシュのエネルギーの値を指定する
  - **Random_Param.Build_Temp** ··· **Num_Torsion_State = 2** or **3** の場合、2 面角を発生させる温度。
    Delta_E と **Build_Temp** を用いてボルツマン因子より確率を計算
  - **Random_Param.Scale_Param** ··· MD atom/SCF segment の比
  - **Helix_Param.Density** ··· 結晶部の密度
  - **Helix_Param.Torsion[]** ··· らせん構造を構築する場合の二面角。トランス=0。複数の二面角の連続によるらせんの場合、二面角が出現する順に値を配列データとして持つ。
  - **Helix_Param.Helix_Length** ··· 1atom あたりの helix の長さ (概算値)
    結晶相の分子鎖密度を見積もるために概算値を設定しておく

- **Crystal**
  - **UDF_Name** ··· 結晶構造データを持つ UDF 名
  - **Num_X_Cell** ··· 単位格子を x 軸方向に繰り返す数
  - **Num_Y_Cell** ··· 単位格子を y 軸方向に繰り返す数
  - **Num_Z_Cell** ··· 単位格子を z 軸方向に繰り返す数
  - **Unit_Length** ··· 結晶構造データで指定されるセルサイズと、実際にシミュレーションで用いる長さの単位との換算値
    例えば、結晶構造データがオングストローム (Å) の様な実単位を持つ値で記述されている場合、**COGNAC** に用いる長さの値として reduced length L = real length(Å)/Unit_Length となるように値をセットする

Crystal の場合は単位格子データを UDF より読み込み、そのサイズを **Num_X/Y/Z_Cell** 倍するので、ユニットセルサイズはインプットで指定するサイズに上書きされる。

Relaxation

初期構造の緩和の設定

- **Relaxation** ··· 初期構造のエネルギー緩和を行うかどうかのフラグ。初期構造を読み込み、あるいは発生した後に起動される
- **Method** ··· 初期構造緩和のアルゴリズム [Keyword]DYNAMICS/MINIMIZE
- **Max_Relax_Force** ··· 初期構造のエネルギー緩和を行う際、原子にかかる力をカットして発散しないようにする値の初期値
- **Max_Relax_Steps** ··· 緩和ステップの最大値

5.2.4 Molecular_Attributes

シミュレーションに用いる、Atom および Interaction Site のタイプ、Bond,Angle および Torsion ポテンシャルに関するデータを定義する。計算に用いるタイプ／ポテンシャルの数に応じて各々配列としてデータを持つ。

Atom_Type[]

- **Name** ··· 原子種の名前。（重複しない限り）任意に付けられる
- **Mass** ··· 原子種の質量

Bond_Potential[]

- **Name** ··· ボンドポテンシャルの名前。（重複しない限り）任意に付けられる
- **Potential_Type** ··· ポンドポテンシャルのタイプ [Keyword]

  ```
  Harmonic
  FENE_LJ
  Gauss
  Morse
  Bond_Polynomial
  DPD
  Table_Bond
  User_Bond
  ```

  個々のタイプの内容に関しては §2.6.1 参照
- **R0** ··· 平衡結合長

  以下に Bond potential のタイプキーワードに応じて必要なパラメータを解説する

  - **Harmonic**
    - **K** ··· バネ定数

- **FENE_LJ**
  - **R_max** ··· FENE 部分の伸びきり長
  - **K** ··· FENE 部分のバネ定数
  - **sigma** ··· LJ 部分の径
  - **epsilon** ··· LJ 部分のエネルギー

- **Gauss**
  - **Temperature** ··· ポテンシャルを計算する温度。シミュレーション温度と違う値を与えることもプログラム上は可能

- **Morse**
  - **A** ··· 2.47 式あるいは 2.48 で用いられるパラメータ A
  - **B** ··· 2.47 式あるいは 2.48 で用いられるパラメータ B
  - **Zero_at_Infinity** ··· **NO** の場合 2.47 式が、**YES** の場合 2.48 式が用いられる

- **Bond_Polynomial**
  - **N** ··· 多項式展開の次数+1
  - **p[]** ··· 各次数における係数 p[0] - p[N-1] の配列

- **DPD**
  - **C** ··· バネ定数

- **Table_Bond**
  - **UDF_Name** ··· テーブル UDF 名
  - **Use_Fast** ··· 高速化されたテーブルポテンシャルを用いる場合 **YES** を選択。
    YES の場合は **Method** でエネルギー値を得る方法を選択。
    - **Interpolation** ··· 線形内挿により値を得る
    - **Nearest_Value** ··· 最も近いテーブル値をそのまま用いる。

    NO の場合は **Order** にテーブルから内挿してエネルギー、力を計算する際の内挿次数を入力する

- **User_Bond**
  - **Index** ··· 使用する **User_Bond** の Index
  - **Parameter[].Name** ··· パラメータの名前（重複しない限り任意に付けられる）
  - **Parameter[].Value** ··· パラメータの値

Angle_Potential[]

- **Name** ··· アングルポテンシャルの名前。（重複しない限り）任意に付けられる
- **Potential_Type** ··· アングルポテンシャルのタイプ [Keyword]

```
Theta
Theta2
Cosine
Theta_Polynomial
Table_Angle
User_Angle
```

個々のタイプの内容に関しては §2.6.2 参照

- **theta0** ··· 平衡角（Degree 単位での値を入力）

以下に Angle potential のタイプキーワードに応じて必要なパラメータを解説する

- **Theta/Theta2/Cosine**
  - **K** ··· バネ定数（**Theta** タイプポテンシャルの場合、ユニットは ϵ/rad^2 となる。その他の場合は ϵ である）
- **Theta_Polynomial**
  - **N** ··· 多項式展開の次数+1
  - **p[]** ··· 各次数における係数 p[0] - p[N-1] の配列
- **Table_Angle**
  - **UDF_Name** ··· テーブル UDF 名
  - **Use_Fast** ··· 高速化されたテーブルポテンシャルを用いる場合 **YES** を選択。
    YES の場合は **Method** でエネルギー値を得る方法を選択。
    - **Interpolation** ··· 線形内挿により値を得る
    - **Nearest_Value** ··· 最も近いテーブル値をそのまま用いる。

    NO の場合は **Order** にテーブルから内挿してエネルギー、力を計算する際の内挿次数を入力する
- **User_Angle**
  - **Index** ··· 使用する **User_Angle** の Index
  - **Parameter[].Name** ··· パラメータの名前（重複しない限り任意に付けられる）
  - **Parameter[].Value** ··· パラメータの値

Torsion_Potential[]

- **Name** ··· 二面角ポテンシャルの名前。（重複しない限り）任意に付けられる
- **Potential_Type** ··· 二面角ポテンシャルのタイプ [Keyword]

```
Cosine_Polynomial
Amber
Dreiding
Table_Torsion
User_Torsion
```

個々のタイプの内容に関しては §2.6.3 参照

以下に Torsion potential のタイプキーワードに応じて必要なパラメータを解説する

- **Cosine_Polynomial**
  - **K** ··· 多項式展開のすべての項にかかる定数
  - **N** ··· 多項式展開の次数+1
  - **p[]** ··· 各次数における係数 p[0] - p[N-1] の配列

- **Amber**
  - **PK** ··· エネルギー障壁の 1/2
  - **IDIVF** ··· ポテンシャルを与える二面角に定義される **Torsion_Potential** の数
  - **PN** ··· ポテンシャルの周期。たとえば 180 ℃で　1 周期であれば **PN**=2
  - **PHASE** ··· 位相角。0 ℃のポテンシャルが最大であれば　**PHASE**=0
  - **trans_is_0** ··· オリジナルの Amber 力場に対応して、二面角においてシスを 0 ℃と定義する場合は **trans_is_0**=0 とする

- **Dreiding**
  - **V** ··· エネルギー障壁
  - **phi0** ··· 平衡角。0 ℃のポテンシャルが最小であれば　**phi0**=0
  - **n** ··· ポテンシャルの周期。たとえば 180 ℃で　1 周期であれば **n**=2
  - **trans_is_0** ··· オリジナルの Dreiding 力場に対応して、二面角においてシスを 0 ℃と定義する場合は **trans_is_0**=0 とする

- **Table_Torsion**
  - **UDF_Name** ··· テーブル UDF 名
  - **Use_Fast** ··· 高速化されたテーブルポテンシャルを用いる場合 **YES** を選択。
    YES の場合は **Method** でエネルギー値を得る方法を選択。
    - **Interpolation** ··· 線形内挿により値を得る
    - **Nearest_Value** ··· 最も近いテーブル値をそのまま用いる。

    NO の場合は **Order** にテーブルから内挿してエネルギー、力を計算する際の内挿次数を入力する
- **User_Torsion**
  - **Index** ··· 使用する User_Torsion の Index。
  - **Parameter[].Name** ··· パラメータの名前（重複しない限り任意に付けられる）
  - **Parameter[].Value** ··· パラメータの値

Interaction_Site_Type[]

- **Name** ··· Interaction Site の名前。（重複しない限り）任意に付けられる
- **Num_of_Atoms** ··· Site を定義する Atom の数

- **Range** ··· Neighbour list をつくる際、リストアップする最大半径。

  2 種類の **Interaction_Site** の **Range** の和以下の距離の場合、Neighbour list に登録される。
  通常 $\mathrm{Range}_{\mathrm{site1}} + \mathrm{Range}_{\mathrm{site2}} = cutoff + \alpha$ になるように設定。α は Neighbour list に登録する距離と、実際に相互作用を計算するカットオフ距離との間のバッファー距離で、任意の長さを与えることができる。α を大きくすると Neighbour list の更新の頻度が少なくなるが、毎回の Non bonding interaction を計算する Neighbour list の数が多くなる。両者のバランスを確認しながら計算効率が最大になる、距離を選ぶのがよい。
  外場のみを作用させるサイトの場合は $Range = 0.0$ に設定すると Neighbour list には登録されない

Electrostatic_Site_Type[]

- **Name** ··· Electrostatic site type の名前。（重複しない限り）任意に付けられる
- **Type_Name** ··· Electrostatic site type のタイプ名 [Keyword]

  ```
  POINT_CHARGE
  CENTER_DIPOLE
  END_DIPOLE
  ```

- **Polarizability** ··· Electrostatic site type の分極率。Field electrostaitic 法の場合に用いられる。

5.2.5 Interactions

シミュレーションに用いる、相互作用ポテンシャルに関するデータを定義する。計算に用いるポテンシャルの数に応じて各々配列としてデータを持つ。

Pair_Interaction[]

- **Name** ··· Pair interaction の名前。（重複しない限り）任意に付けられる
- **Potential_Type** ··· Pair interaction のタイプ [Keyword]

  ```
  Lennard_Jones
  Lennard_Jones_EV
  General_Lennard_Jones
  Gay_Berne
  GB_LJ
  DPD
  Morse
  Buckinhgam
  Table_Pair_Potential
  User_Pair_Interaction
  ```

  個々のタイプの内容、およびパラメータの意味に関しては §2.6.4 参照

- **Site1_Name/Site2_Name** ··· Pair interaction を作用させる、対となる Interaction site の名前

- **Cutoff** ··· Cut off 距離
- **Scale_1_4_Pair** ··· 1-4 atom 間の Pair Interaction のスケールファクター。Torsion potential を計算し、かつ" Simulation_Conditions.Calc_Potential_Flags.Non_Bonding_1_4" において 1-4 pair interaction を計算する設定の場合に有効

  以下に Pair interaction のタイプキーワードに応じて必要なパラメータを解説する

  - **Lennard_Jones**
    - **sigma** ··· Lennard Jones 径。 2.58 式参照
    - **epsilon** ··· Lennard Jones。 エネルギー 2.58 式参照
  - **Lennard_Jones_EV**
    - **sigma** ··· Lennard Jones 径。2.59 式参照
    - **epsilon** ··· Lennard Jones。 エネルギー 2.59 式参照
    - **R_EV** ··· Excluded volume の直径　2.59 式参照
  - **General_Lennard_Jones**
    - **sigma** ··· Lennard Jones 径。 2.60 式参照
    - **epsilon** ··· Lennard Jones。 エネルギー 2.60 式参照
    - **A** ··· 反発項の係数。
    - **B** ··· 分散項の係数。
    - **m** ··· 反発項の次数。2.60 式参照
    - **n** ··· 分散項の次数。2.60 式参照
  - **Gay_Berne**
    - **sigma** ··· サイズパラメータ σ_0。2.61,2.62 式参照
    - **epsilon** ··· エネルギーパラメータ ϵ。2.66 式参照
    - **l1,l2** ···　2.64,2.65 式 l_i, l_j
    - **d1,d2** ··· 2.63,2.64,2.65 式 d_i, d_j
    - **mu,nu** ···　2.66 式 μ, ν
    - **alpha2,k2** ···　2.68 式 α', 2.69 式 k'
  - **GB_LJ**
    - **sigma** ··· サイズパラメータ σ_0。2.70,2.71 式参照
    - **epsilon** ··· エネルギーパラメータ ϵ。2.72 式参照
    - **l2** ···　2.75 式 l_j
    - **d1,d2** ··· 2.74,2.75 式 d_i, d_j
    - **k2** ···　2.75 式 k'
    - **mu** ···　2.72,2.75 式 μ
  - **DPD**

    - **a** ··· 相互作用パラメータ。2.76 式参照
    - **gamma** ··· Friction constant。2.34 式参照。

- **Morse**
    - **A** ··· 2.77 式の A
    - **B** ··· 2.77 式の B
    - **r0** ··· 2.77 式の r_0

-
    - **A** ··· 2.78 式の A
    - **B** ··· 2.78 式の B
    - **C** ··· 2.78 式の C
- **Table_Pair_Potential**
    - **UDF_Name** ··· テーブル UDF 名
    - **Use_Fast** ··· 高速化されたテーブルポテンシャルを用いる場合 **YES** を選択。
      YES の場合は **Method** でエネルギー値を得る方法を選択。
        - **Interpolation** ··· 線形内挿により値を得る
        - **Nearest_Value** ··· 最も近いテーブル値をそのまま用いる。

      NO の場合は **Order** にテーブルから内挿してエネルギー、力を計算する際の内挿次数を入力する
- **User_Pair_Interaction**
    - **Index** ··· 使用する User pair interaction の Index。
    - **Parameter[].Name** ··· パラメータの名前（重複しない限り任意に付けられる）
    - **Parameter[].Value** ··· パラメータの値

External_Interaction[]

- **Name** ··· External interaction の名前。（重複しない限り）任意に付けられる
- **Potential_Type** ··· External interaction のタイプ [Keyword]

```
LJ_Wall
LJ_Atomic_Wall
Static_Field
Density_Field
Velocity_Field
Total_Density_Constrain
External_Angle
External_Torsion
User_External_Field
```

個々のタイプの内容に関しては §2.6.5 参照

- **Site_Name** ··· External interaction を作用させる Interaction_Site の名前

  以下に External interaction のタイプキーワードに応じて必要なパラメータを解説する

  - **LJ_Wall** : フラットな壁のポテンシャル
    - **Cutoff** ··· Cut off 距離
    - **sigma** ··· Lennard Jones 径
    - **epsilon** ··· Lennard Jones エネルギー
    - **Density** ··· 表面密度
      表面上の面密度で指定する
    - **Direction** ··· 壁を置く位置 [Keyword]
      z と入力すると z=0 および z=max の位置の xy 平面上に壁をおく。また zl あるいは zu と指定すると、z=0 および z=max の片面にのみポテンシャルが作用される。同様に x(l/u),y(l/u) も可

  - **LJ_Atomic_Wall** : 構造を持つ壁のポテンシャル
    - **Cutoff** ··· Cut off 距離
    - **sigma** ··· Lennard Jones 径
    - **epsilon** ··· Lennard Jones エネルギー
    - **Density** ··· 表面密度
      表面上の面密度で指定する
    - **Position** ··· 壁を置く位置。**LJ_Atomic_Wall** の場合、z=0 および z=max の xy 平面上あるいはそのどちらかに壁を置くことが出来る [Keyword]

      ```
      Both_Side
      Lower_Side
      Upper_Side
      ```

    - **shear** ··· ずり速度 $\dot{\gamma}_{xz}$
      LJ_Atomic_Shear の場合は、xy 平面のポテンシャルのみをサポートしている。

  - **Staic_Field**
    - **Field** ··· Field の値。3 次元のベクトル値

  - **Density_Field** : 濃度場ポテンシャル
    - **UDF_name** ··· 濃度場データを持つ **SUSHI**、**Muffin_phaseseparation** あるいは **Gird_Density** データを持つ **COGNAC** の UDF ファイル名。**UDF_name** の指定がない場合は、**Density_Output** で指定する条件で計算される、濃度場が用いられる。
    - **Component_Index** ··· ポテンシャルを計算する濃度場に対応する **SUSHI** 出力 UDF 内の **SUSHIOutput.phi.value[].comp[]** の Index。**Muffin_phaseseparation** の場合は **Name** に **VolumeFraction** が指定されている field の **field.scalar_field[].value[].comp[]** の Index。**COGNAC** の場合は、**Grid_Density.phi.value[].comp[]** の Index。
    - **Potential_Type** ··· ポテンシャルタイプ [Keyword]
      ポテンシャルの詳細は §2.6.5 参照。

```
Density_Biased_Potential
Lennard_Jones
Reciprocal_Power
```

· **Density_Biased_Potential** : Density biased potential のパラメータ。
chi ··· χ parameter
· **Lennard_Jones :** Lennard Jones タイプポテンシャルのパラメータ。
sigma ··· Lennard Jones 径
epsilon ··· Lennard Joned エネルギー
· **Reciprocal_Power :** Reciprocal power タイプポテンシャルのパラメータ。
Order ··· ポテンシャルの次数
Coefficient ··· 係数

– **Total_Density_Constrain** : 全濃度の拘束ポテンシャル
∗ **coeff** ··· ポテンシャルの係数

– **External_Angle** : 指定された領域内に存在する Three atoms site に Cosine タイプの結合角ポテンシャルを作用させる
∗ **theta0** ··· 平衡角
∗ **K** ··· バネ定数
∗ **Min_Position** ··· 領域指定の最小値 (x,y,z)
∗ **Max_Position** ··· 領域指定の最大値 (x,y,z)

– **External_Torsion** : 指定された領域内に存在する Four atoms site に Cosine Polynomial タイプの二面角ポテンシャルを作用させる
∗ **phi0** ··· 二面角ポテンシャルのシフト角
∗ **K** ··· 多項式展開のすべての項にかかる定数
∗ **N** ··· 多項式展開の次数+1
∗ **p[]** ··· 各次数における係数 p[0] - p[N-1] の配列
∗ **Min_Position** ··· 領域指定の最小値 (x,y,z)
∗ **Max_Position** ··· 領域指定の最大値 (x,y,z)

– **Tethered_Force**
∗ **K** ··· ばね定数
– **User_External_Field**
∗ **Index** ··· 使用する User_External_Field の Index
∗ **Parameter[].Name** ··· パラメータの名前（重複しない限り任意に付けられる）
∗ **Parameter[].Value** ··· パラメータの値

Electrostatic_Interacton[]

- **Name** ··· Electrostatic interaction の名前。（重複しない限り）任意に付けられる
- **Algorithm** ··· Electrostatic interaction 計算アルゴリズム [Keyword]

  ```
  Cutoff_Coulomb
  Cutoff_Coulomb_Debye
  Reaction_Field
  Ewald
  Field_Electrostatic
  PPPM
  ```

  個々のタイプの内容に関しては §2.6.6 参照

- **Scale_1_4_Pair** ··· 1-4 atom 間の Electrostatic Interaction のスケールファクター。点電荷の場合に有効。ただし **Field_Electrostatic** をにおいてはサポートされていない。また Torsion potential を計算し、かつ” Simulation_Conditions.Calc_Potential_Flags.Non_Bonding_1_4” において 1-4 pair interaction を計算する設定の場合に有効。

  以下に Coulomb interaction のタイプキーワードに応じて必要なパラメータを解説する

  - **Cutoff_Coulomb**
    - **Dielectric_Constant** ··· 系全体の比誘電率
    - **cutoff** ··· カットオフ距離
  - **Cutoff_Coulomb_Debye**
    - **Dielectric_Constant** ··· 系全体の比誘電率
    - **cutoff** ··· カットオフ距離
    - **kappa** ··· Debye length の逆数
  - **Reaction_Field**
    - **Dielectric_Constant** ··· カットオフ距離以遠の周囲の媒体の比誘電率。この値が 1.0 未満であれば、周囲の媒体の比誘電率を ∞ として扱う。
    - **cutoff** ··· カットオフ距離
  - **Ewald**
    - **Dielectric_Constant** ··· カットオフ距離以遠の周囲の媒体の比誘電率。この値が 1.0 未満であれば、周囲の媒体の比誘電率を ∞ として扱う。
    - **R_cutoff** ··· 実空間項のカットオフ距離
    - **Ewald_Parameters** ··· Ewald 法に関するパラメータの設定法 [Keyword]

      ```
      Auto
      Manual
      ```

· **Auto** ··· パラメータをプログラムが自動的に設定する
· **Manual** ··· パラメータをマニュアルで設定する

alpha ··· 電荷分布の広がりを表すパラメータ

（注意） バージョン 6.0 以前のパラメータ **al** と **alpha** は以下の関係となっている。
al = **alpha** * (unit cell length)
バージョン 6.1 より Ewald 法において直方体ユニットセルをサポートするために **alpha** をパラメータとして入力するように変更された。

K_cutoff ··· 波数空間のカットオフの設定。**nh,nk,nl** と異方的に設定することが可能

– **Field_Electrostatic**[39]
  * **Dielectric_Constant** ··· 系全体の比誘電率
  * **gamma** ··· 電荷のスケールパラメータ
  * **range** ··· Charge smearing の範囲
  * **zeta** ··· Poisson 方程式を解く際の緩和係数。通常は **0.15** 程度
  * **error** ··· Poisson 方程式を解く際の許容誤差。通常は **0.03** 程度
  * **max_iteration** ··· Poisson 方程式を解く際の最大 iteration 数

– **PPPM**[40, 41]
  * **Dielectric_Constant** ··· 系全体の比誘電率
  * **R_cutoff** ··· 実空間項のカットオフ距離
  * **PPPM_Parameters** ··· Ewald 法に関するパラメータの設定法 [Keyword]

```
Auto
Manual
```

· **Auto** ··· パラメータをプログラムが自動的に設定する
· **Manual** ··· パラメータをマニュアルで設定する

alpha ··· 電荷分布の広がりを表すパラメータ。Ewald 法と同様

Number_of_Grid ··· Poisson 方程式を解くためのメッシュ数の指定。**nx,ny,nz** で各軸方向の分割数を設定する。

  * **error** ··· Poisson 方程式を解く際の許容誤差。通常は **0.03** 程度
  * **max_iteration** ··· Poisson 方程式を解く際の最大 iteration 数

5.2.6 React_Conditions

化学反応（結合の生成、解離）の設定。

- **React_Flag** ··· 化学反応を取り入れるかどうかのフラグ
  ON/**OFF** で指定
- **Atom_Exchange** ··· Atom Type の置換に関する設定

- **Exchange_Type_Array**[] ··· Atom Type 置換のタイプの設定。設定するタイプ数の配列
  - **Interaval_of_Reaction** ··· Atom Type の置換をチェックするタイムステップのインターバル
  - **Probability** ··· 下記の **Exchang_Type** で指定した、置換の条件を満たした際に、実際に置換を起こす確率。0 – 1.0 の値を入力する。
  - **Target_Atom_Type** ··· 置換の対象の Atom Type
  - **Product_Atom_Type** ··· 置換後の Atom Type
  - **Product_Atom_Name** ··· 置換後の Atom Name。空白の場合は Atom Name は変更されない
  - **Product_Interaction_Site_Type** ··· 置換後の Interaction Site Type
  - **Exchange_Type** ··· Atom Type 置換判定の設定
    - **Type_Keyword** ··· 判定のタイプを指定するキーワード。現在は **Region** のみサポート。
    - **Region** ··· 指定した空間領域に存在する Atom を置換する。境界条件によりユニットセル内に存在するイメージの座標により判定される。

      Min_Position ··· 空間領域の最小の座標値。**x,y,z** 座標を指定する。
      Max_Position ··· 空間領域の最大の座標値。**x,y,z** 座標を指定する。

      Min_Position,Max_Position とも **0** と指定すると、その軸はユニットセルすべての領域が指定される。

- **Bond_Creation** ··· 結合生成に関する設定
  - **Reactive_Atom**[] ··· 反応に関わる Atom の設定。反応可能な Atom_Type の数の配列
    - **Atom_Type_Name** ··· 反応しうる Atom type。**Molecular_Attribute** で定義されている **Atom_Type** の **Name**
    - **Max_bond_Num** ··· 指定した Atom type がとり得る最大結合数
  - **Creation_Type_Array**[] ··· 結合生成の設定。設定する結合種の数の配列
    - **Interval_of_Reaction** ··· 結合生成をチェックするタイムステップのインターバル
    - **Probability** ··· 下記の **Creation_Type** で指定した条件を満たした場合に実際に結合を生成する確立。0 – 1.0 の値を入力する
    - **In_Chain** ··· 分子内反応を許すかどうかのフラグ
      ON/OFF で指定
    - **Creation_Type** ··· 結合生成を判定する条件の設定
      - **Type_Keyword** ··· 結合生成を判定する条件を選択
        Simple_Creation および **Polymerization** をサポート
      - **Simple_Creation** ··· **Simple_Creation** の諸設定
        Threshold_Distance ··· 結合生成のしきい距離　**Potential_Name** ··· 新たに生成する bond の Bond potential。**Molecular_Attribute** で定義されている **Bond_Potential** の **Name**
        Atom_Type_Sequence ··· 反応する **Atom_Type_Name** の組み合わせを **atom1/atom2** で指定。**atom2** が空欄の場合は **atom1** で指定した **Atom_Type** 同士が反応する

· **Polymerization** ··· **Polymerization** の諸設定

Threshold_Distance ··· 結合生成のしきい距離 **Potential_Name** ··· 新たに生成する bond の Bond potential。**Molecular_Attribute** で定義されている **Bond_Potential** の **Name**

Atom_Type_Sequence ··· 反応前後の Atom Type の指定

Reactive_Atom··· 反応活性種の Atom Type

Monomer··· **Reactive_Atom** と反応する Atom Type

New_Reactive_Atom··· **Monomer** が反応して新たに生成する活性種の Atom Type

New_Inactive_Atom··· **Reactive_Atom** が反応して新たに生成する不活性種の Atom Type

Atom_Name_Sequence ··· 反応後の Atom Name の指定。空欄の場合は変更されない

New_Reactive_Atom··· **Monomer** が反応して新たに生成する活性種の Atom Name

New_Inactive_Atom··· **Reactive_Atom** が反応して新たに生成する不活性種の Atom Name

Maximum_Chain_Length··· 最大鎖長。0 の場合は制約なし

– **Potential_Assignment** ··· 結合生成時に新たに定義される Angle/Torsion の Potential に関する設定

∗ **Angle**[] ··· 新たに定義される Angle potential に関する設定の配列

· **Potential_Name** ··· 新たに生成する angle の Angle potential name
Molecular_Attribute で定義されている **Angle_Potentia** の **Name**

· **Atom_Type_Sequence** ··· 新たに生成する angle を **Atom_Type_Name atom1/atom2/atom3** で指定する

· **Bond_Num_of_atom2** ··· 新たに生成する angle の中央の Atom, **atom2** が持つ結合の数

生成する結合を含めた結合数がこの値に等しくなる時のみ、Potential name がアサインされる

∗ **Torsion**[] ··· 新たに定義される Torsion potential に関する設定の配列

· **Potential_Name** ··· 新たに生成する torsion の Torsion potential
Molecular_Attribute で定義されている **Torsion_Potential** の **Name**

· **Atom_Type_Sequence** ··· 新たに生成する torsion を **Atom_Type_Name atom1/atom2/atom3/atom4** の順で指定する

· **Bond_Num_of_atom2** ··· 新たに生成する torsion の中央部の Atom, **atom2** が持つ結合の数

· **Bond_Num_of_atom3** ··· 新たに生成する torsion の中央部の Atom, **atom3** が持つ結合の数
生成する結合を含めた結合数が上の 2 つの値に等しくなる時のみ、Potential name がアサインされる

• **Bond_Scisson** ··· 結合解離に関する設定

– **Bond_Scission**[] ··· 解離する bond の設定。解離しうる Bond potential の配列

* **Bond_Potential_Name** ··· 解離する Bond potential name
  Molecular_Attribute で定義されている **Bond_Potential** の **Name**
* **Scission_Type** ··· 解離を判定する方法。**Length** および **Region** をサポート
* **Length** ··· **Scission_Type = "Length"** の際のパラメータ
  · **Scission_Length** ··· 結合解離のしきい距離

* **Region** ··· **Scission_Type = "Region"** の際のパラメータ
  · **Min_Position** ··· 矩形領域の最小座標。**x,y,z** で指定
  · **Max_Position** ··· 矩形領域の最大座標。**x,y,z** で指定

5.2.7 Set_of_Molecules

Set_of_Molecules は以下の階層構造を持つ。

```
Set_of_Molecules---molecule[]---atom[]
                              |-bond[]
                              |-angle[]
                              |-torsion[]
                              |-interaction_Site[]
                              |-electrostatic_Site[]
```

以下に **molecule,atom,bond,angle,torsion,interaction_Site** および **electrostatic_Site** の持つパラメータに関して解説する

molecule[]

- **Mol_Name** ··· 分子の名前
  任意に付けられるが、同じ種類の分子を複数持つ場合は同名を付けることを原則とする

atom[]

- **Atom_ID** ··· 原子の ID
  任意に付けられるが基本的に重複しないように与える
- **Atom_Name** ··· 原子の名前。任意に付けられる
- **Atom_Type_Name** ··· **Molecular_Attributes** 内で定義されている **Atom_Type** の名前。個々の atom に type を設定する
- **Chirality** ··· キラリティーの設定 **1** (R or S)/**0**(random) /**-1**(S or R)
  （注）R/S に関する正確な定義は行っていない。1 – 1 あるいは-1 – -1 のペアの場合はメソ、1 – -1 のペアの場合はラセミという相対的な指定となる。
- **Main_Chain** ··· Helix の作成などの場合に主鎖を定義する必要がある場合、主鎖の Atom を **true(1)** にセットする

- **Attributes**[] ··· 原子の属性。要素に **Name**, **Value** を持ち、任意の名前の属性を定義して値を与えることが出来る。例として、原子の属するモノマーユニット名などを与え、描画、ズーミングなどに利用する。**COGNAC** 本体はこのオブジェクトは利用しない

bond[]

- **Potential_Name** ··· **Molecular_Attributes** 内で定義されている **Bond_Potenital** の名前。個々の **bond** に Potential を設定する
- **atom1/atom2** ··· **bond** を構成する **atom** の Index。対象の **atom** は当然同一 **molecule** 内に存在するので、**atom** の配列 Index のみを指定する
- **Order** ··· 結合次数。主に描画や力場パラメータのセットのために用いられ、**COGNAC** 本体は利用しない

angle[]

- **Potential_Name** ··· **Molecular_Attributes** 内で定義されている **Angle_Potenital** の名前。個々の **angle** に Potential を設定する
- **atom1/atom2/atom3** ··· **angle** を構成する **atom** の Index。対象の **atom** は当然同一 **molecule** 内に存在するので、**atom** の配列 Index のみを指定する

torsion[]

- **Potential_Name** ··· **Molecular_Attributes** 内で定義されている **Torsion_Potenital** の名前。個々の **torsion** に Potential を設定する
- **atom1/atom2/atom3/atom4** ··· **torsion** を構成する **atom** の Index。対象の **atom** は当然同一 **molecule** 内に存在するので、**atom** の配列 Index のみを指定する

interaction_Site[]

- **Type_Name** ··· **Molecular_Attributes** 内で定義されている **Interaction_Site_Type** の名前。個々の **interaction_Site** に定義されている Type を与える
- **atom**[] ··· **interaction_Site** を構成する **atom** の Index。**interaction_Site** を定義する **atom** の数を要素数とする配列。対象の **atom** は当然同一 **molecule** 内に存在するので、**atom** の配列 Index のみを指定する

electrostatic_Site[]

- **Type_Name** ··· Electrostatic interaction を作用させるサイトのタイプ 。以下のキーワードを与えると、タイプが規定される。

```
POINT_CHARGE
CENTER_DIPOLE
END_DIPOLE
```

あるいは、Field electrostatic 法を用いる際は、**Electrostatic_Site_Type**[] に定義されている。**Name** を指定することによりタイプを規定することが出来る。

- **ES_Element** ··· **electrostatic_Site** の持つ電荷
  Type_Name により、**POINT_CHARGE** の場合は点電荷、**DIPOLE** の場合は Dipole moment を表す
- **atom**[] ··· **electrostatic_Site** を構成する **atom** の Index。**POINT_CHARGE** の場合は要素数 1、**DIPOLE** の場合は要素数 2。対象の **atom** は当然同一 **molecule** 内に存在するので、**atom** の配列 Index のみを指定

5.2.8 Structure

Structure は以下の階層構造を持つ。

```
Structure---Position---mol[]---atom[]
          |-Velocity---mol[]---atom[]
          |-Force    ---mol[]---atom[]
          |-Unit_Cell
```

- **Position** ··· Atom の位置
- **Velocity** ··· Atom の速度
- **Force** ··· Atom にかかる力

  Position,Velocity,Force 以下の **mol[].atom[]** は **Set_of_Molecules** の **molecule[].atom[]** に対応し、同じ配列 Index で指定する
- **Unit_Cell** ··· Unit Cell の情報
  - **Density** ··· 密度
  - **Shear_Strain** ··· Lees-Edwards 境界条件の場合の shear strain の値
  - **Cell_Size** ··· ユニットセルサイズ (**a,b,c,alpha,beta,gamma**)

5.2.9 Unit_Parameter

Unit_Parameter は以下のパラメータを持つ

- **Name** ··· ユニットセットの名前
- **Comment** ··· コメント
- **Mass** ··· Reduced mass
  オリジナルデータは [amu](atomic mass unit) の単位を持つ。ただし、**GOURMET** 上では任意の単位で入出力を行うことが可能。
- **Enregy** ··· Reduced energy
  オリジナルデータは [kJ/mol] 単位を持つ。ただし、**GOURMET** 上では任意の単位で入出力を行うことが可能。

- **Length** ··· Reduces length
  オリジナルデータは [nm] の単位を持つ。ただし、**GOURMET** 上では任意の単位で入出力を行うことが可能。

5.2.10 Draw_Attributes

Draw_Attributes は以下のパラメータを持つ。

- **Atom_Type**[] ··· 属性を定義する **Atom_Type** の配列
  - **Name** ··· **Atom_Type** の名前
  - **color** ··· 描画時の色の指定。Select 機能で表示される色のリストより選択
  - **transparency** ··· 描画時の透過度の設定。0.0-1.0 の値をとる。(1.0:不透過、0.0:透明)
  - **radius** ··· Atom を Ball で表示する際の半径
- **Bond_Potential**[] ··· 属性を定義する **Bond_Potential** の配列
  - **Name** ··· **Bond_Potential** の名前
  - **color** ··· 描画時の色の指定。Select 機能で表示される色のリストより選択
  - **transparency** ··· 描画時の透過度の設定。0.0-1.0 の値をとる。(1.0:不透過、0.0:透明)
  - **radius** ··· Bond を Stick/Rod で表示する際の半径
- **Molecule**[] ··· 属性を定義する **Molecule** の配列
  - **Name** ··· **Molecule** の名前
  - **color** ··· 描画時の色の指定。Select 機能で表示される色のリストより選択
  - **transparency** ··· 描画時の透過度の設定。0.0-1.0 の値をとる。(1.0:不透過、0.0:透明)
  - **radius** ··· Ball-Stick で表示する際の Ball の半径

5.3 出力 UDF 解説

出力 UDF には以下の項目が出力される

- 入力パラメータ
  Initial record に入力 UDF で指定したパラメータがエコーバックされる

- **Statistics_Data**

  計算結果諸量。表 5.10 に **Statistics_Data** 以下のパラメータリストを記す。各パラメータの詳細については本文を参照のこと。

  以下の諸量の瞬間値 (**Instantaneous**)、区間平均 (**Batch_Average**)、全平均 (**Total_Average**) が **Statistics_Data** 以下に出力される。ただし、**Energy_Flow** に関しては区間積算値、全積算値が出力される。(§2.3.5 参照)

  - **Energy** ··· エネルギー **Energy** の項では、以下の項目に分割されて出力される
    Hamiltonian[*1] / **Total Energy** / **Kinetic Energy** / **Potential Energy** / **Bond Energy** / **Angle Energy** / **Torsion Energy** / **Non Bonding Energy** / **Couomb Energy** / **External Energy**

    *1 ハミルトニアンが定義できないアルゴリズムでは Total Energy に等しい値が出力される
  - **Temperature** ··· 温度
  - **Pressure** ··· 圧力
  - **Stress**
    - **Total** ··· 通常のストレステンソル。Bond, Non bond および Kinetic 項すべての和
    - **Bond** ··· Bond 項のみから計算されるストレステンソル
    - **Non_Bond** ··· Non bond 項のみから計算されるストレステンソル
  - **Volume** ··· 体積
  - **Density** ··· 密度
  - **Cell** ··· セルサイズ
  - **Wall_Press** ··· 壁にかかる圧力 (壁のポテンシャルを設定した場合)
  - **Enery_Flow** ··· 温度スケールにより出入りしたエネルギー

- **Set_of_Molecules**
  入力 UDF と同一構造を持つ。
  化学反応を取り入れたシミュレーションを行った場合 Record data に出力される。それ以外は Global record に書き込まれるのみ。

- **Structure**
  Simulation_Condisitons.Output_Flags.Structure で指定された場合、毎レコードに出力される。
  入力 UDF と同一構造を持つ。

- **Averaged_Structure**
  Simulation_Condisitons.Output_Flags.Averaged_Structure で指定された場合、毎レコードに出力される。構造は基本的に **Structure**(§5.2.8) と同様だが **Unit_Cell** の情報は持たない。

表 5.10: Statistics_Data

UDF パス名	意味
”Energy”	運動エネルギー、各ポテンシャルエネルギー
”Energy.Instantaneous”	瞬間値
”Energy.Batch_Average”	区間平均
”Energy.Total_Average”	全平均
”Temperature”	温度
”Temperature.Instantaneous”	瞬間値
”Temperature.Batch_Average”	区間平均
”Temperature.Total_Average”	全平均
”Density”	密度
”Density.Instantaneous”	瞬間値
”Density.Batch_Average”	区間平均
”Density.Total_Average”	全平均
”Volume”	体積
”Volume.Instantaneous”	瞬間値
”Volume.Batch_Average”	区間平均
”Volume.Total_Average”	全平均
”Cell_Size”	セルサイズ
”Cell_Size.Instantaneous”	瞬間値
”Cell_Size.Batch_Average”	区間平均
”Cell_Size.Total_Average”	全平均
”Pressure”	圧力
”Pressure.Instantaneous”	瞬間値
”Pressure.Batch_Average”	区間平均
”Pressure.Total_Average”	全平均
”Stress”	ストレステンソル
”Stress.Total.Instantaneous”	トータルストレス（Bond & Non bond & Kinetic 項）瞬間値
”Stress.Total.Batch_Average”	トータルストレス（Bond & Non bond & Kinetic 項）区間平均
”Stress.Total.Total_Average”	トータルストレス（Bond & Non bond & Kinetic 項）全平均
”Stress.Bond.Instantaneous”	Bond ストレス　瞬間値
”Stress.Bond.Batch_Average”	Bond ストレス　区間平均
”Stress.Bond.Total_Average”	Bond ストレス　全平均
”Stress.Non_Bond.Instantaneous”	Non Bonding ストレス　瞬間値
”Stress.Non_Bond.Batch_Average”	Non Bonding ストレス　区間平均
”Stress.Non_Bond.Total_Average”	Non Bonding ストレス　全平均
”Wall_Pressure”	壁にかかる圧力 (壁のポテンシャルを設定した場合)
”Wall_Pressure.Instantaneous”	瞬間値
”Wall_Pressure.Batch_Average”	区間平均
”Wall_Pressure.Total_Average”	全平均
”Energy_Flow”	温度スケールにより出入りしたエネルギー
”Energy_Flow.Batch_Sum”	区間積算値
”Energy_Flow.Total_Sum”	全積算値

- **Grid_Density**
  Simulation_Conditions.Density_Output で密度分布の出力を設定した場合、record data に出力される。
  フォーマットは **SUSHI** で用いられている Mesh & Field のものと類似している。詳細は「SUSHI ユーザーズマニュアル」参照。
  - **mesh** ··· Mesh の定義。**SUSHI** の Regular Mesh の定義に順ずる。
  - **boundary_conditions** ··· 境界条件の定義。**SUSHI** の定義に順ずる。
  - **atom_name**[] ··· **phi.value**[].**comp**[] に出力される濃度場に対応する **Atom_Name** の配列。
  - **phi** ··· 濃度場。形式は **SUSHI** に順ずる。

- **Correlation_Functions**
  Simulation_Conditions.Output_Flags.Correlation_Function で自己相関関数の on the fly 出力を指定した場合、**最終レコード**に結果が出力される。
  - **Stress** ··· Stress の自己相関関数
    xy,yz,zx,xx-yy,yy-zz、および Green-Kubo 公式より得られる緩和弾性率 **G_t** が、時間 **Time** の関数として出力される

- **Unit_Parameter** 単位換算のためのスケーリングパラメータ。入力 UDF と同一構造を持つ

5.4 Table UDF 解説

bond、angle、torsion 等の Table Potential のデータを持つ。この Table UDF のデータ定義は、Mesh & Field を記述する定義に一部したがっている。

- **Mesh**：Mesh の情報（Table Potential の場合、結合長、角度、刻み幅等）を持つ。

  表 5.11 に **Mesh** 以下のパラメータリストを記す。各パラメータの詳細については本文を参照のこと。

表 5.11: Mesh

UDF パス名	意味
”name”	Mesh の名前
”type”	Mesh のタイプ（**COGNAC** では使用されないパラメータ）
”axes[]”	軸のデータ（Mesh の次元に対応する要素数）
”axes[*].values[]”	分割を定義するデータ

  - **name** ··· Mesh の名前
  - **type** ··· Mesh のタイプ。現在 **COGNAC** がサポートするのは Regular のみなのでメッシュタイプの指定には使用していない。
    例外:**type = FORCE** と指定した場合、与えられたテーブルは Potenial table ではなく Force table として解釈される
  - **axes[]**：軸のデータ。Mesh の次元に対応する要素数の配列
    - **values[]**：各軸における、分割を定義するデータ。Regular Mesh の場合以下のデータが入る
      values[0] ··· 最小値
      values[1] ··· 最大値
      values[2] ··· 分割数（100 分割の場合、両端含めて 101 のデータポイントが存在する）

- **FieldValue**：Mesh 上のデータ（Table Potential の場合エネルギー値）を持つ。

  表 5.12 に **FieldValue** 以下のパラメータリストを記す。各パラメータの詳細については本文を参照のこと。

表 5.12: FieldValue

UDF パス名	意味
”value[]”	Mesh 上のデータの配列

  - **value[]** ··· Mesh 上のデータの配列。上記のように (分割数+1)× 次元数の要素を持つ

- **Unit_Parameter**：単位換算のためのスケールパラメータ。入出力 UDF と同一構造

5.5 Crystal UDF 解説

Crystal UDF は以下のデータを持つ。

- **Crystal_Data**：格子の情報を持つ。

  表 5.13 に **Crystal_Data** 以下のパラメータリストを記す。各パラメータの詳細については本文を参照のこと。

表 5.13: Crystal_Data

UDF パス名	意味
"Unit_Cell"	結晶単位格子のサイズ
"Unit_Cell.a"	a 軸長
"Unit_Cell.b"	b 軸長
"Unit_Cell.c"	c 軸長
"Unit_Cell.alpha"	α 角
"Unit_Cell.beta"	β 角
"Unit_Cell.gamma"	γ 角
"Symmetry_Operation"	対称操作の項
"Symmetry_Operation.Operation[]"	対称操作（並進、回転）の配列
"Fractional_Coordinate"	Fractional coordinate のデータ
"Fractional_Coordinate.Coordinate[]"	coordinate のデータ（配列）

  Crystal Data のフォーマットを解説する

  - **Unit_Cell** ··· 結晶単位格子のサイズ
  - **Symmetry_Operation** ··· 対称操作の項。対称操作の数の配列（**Operation**[]）を持つ

    Operation[] のデータ

    * **Invert_Vector** ··· 反転操作のベクトル
    * **Trans_Vector** ··· 並進操作のベクトル

      上記の入力により、asymmetric unit の Fractional coordinate **R** は

      Invert_Vector:**R** + Trans_Vector に変換される
  - **Fractional_Coordinate** ··· Fractional coordinate の値を asymmetric unit の数だけ配列（**Coordinate**[]）として持つ

    Coordinate[] のデータ

    * **x,y,z** ··· asymmetric unit の fractional coordinate

- **Unit_Parameter**：単位換算のためのスケールパラメータ。入出力 UDF と同一構造

第6章　入力データ作成支援ツール –SILK–

ここでは、**COGNAC** 入力データ作製支援ツール **SILK** について解説する。

SILK(Set of molecules Interpreter of Light Kits) とは、**COGNAC** の入力項目である高分子のトポロジーを記述したデータ（**Set_of_Molecules**）を作成するツールである。**Set_of_Molecules** は、広範囲の分子のトポロジーを記述できるフォーマットになっているために、そのデータを直接編集する場合の作業量（特に高分子の場合）は膨大なものになる場合がある。その作業量低減の為に、いくつかの関数群を整備してまとめたものが **SILK** である。

この章では、**GOURMET** の **ACTION** 機能を用いて **Set_of_Molecules** を編集する方法について紹介し、次に、SILK を **GOURMET** の **Python** から実行する方法等について紹介する。

6.1　SILK 操作入門：ACTION による Set_of_Molecules 作成

ACTION を用いた **COGNAC** 入力ファイルの作成手順を以下に示す。

1. Potential map

   SILK 用入力 UDF は、**COGNAC** 入力 UDF とほぼ同じ構造を持つ。雛型として “potential_map.udf” が用意されているので、これを例題として使用する。

   GOURMET を起動し、UDF（ “potential_map.udf”）を Open する。Open 時、画面には Record Number が **0** のデータが表示されている。窓の下のつまみを右へスライドさせ、右下の窓に Record Label である “ACTION_SILK_TEST” が表示されるレコード（Record **12**）であることを確認する。

2. 分子構造の作成

   GOURMET 画面の左側にはエクスプローラー様のデータ構造が表示されている。
   このエリアの **Set_of_Molecules** 上にカーソルを置き、右ボタンクリックするとアクションのメニューがポップアップされる。各メニューについて説明する。

   - **SILK_CREATE_CombPolymer_Bead_Spring_2_branched**
     分岐点が二つ存在する、ビーズスプリングモデルの櫛形高分子（例 : **A10(B5)-A10(B5)-A10**）を作成する。

   - **SILK_CREATE_CombPolymer_Bead_Spring_X_branched**
     任意の数、長さ、原子種のブロックからなる、ビーズスプリングモデルの櫛形高分子（例 : **A10(B5)-A5(C5)-A7(B3)-A10**）を作成する。

   - **SILK_CREATE_LinearPolymer_Bead_Spring_1_Homo**
     ビーズスプリングモデルの線状高分子（例 : **A10**）を作成する。

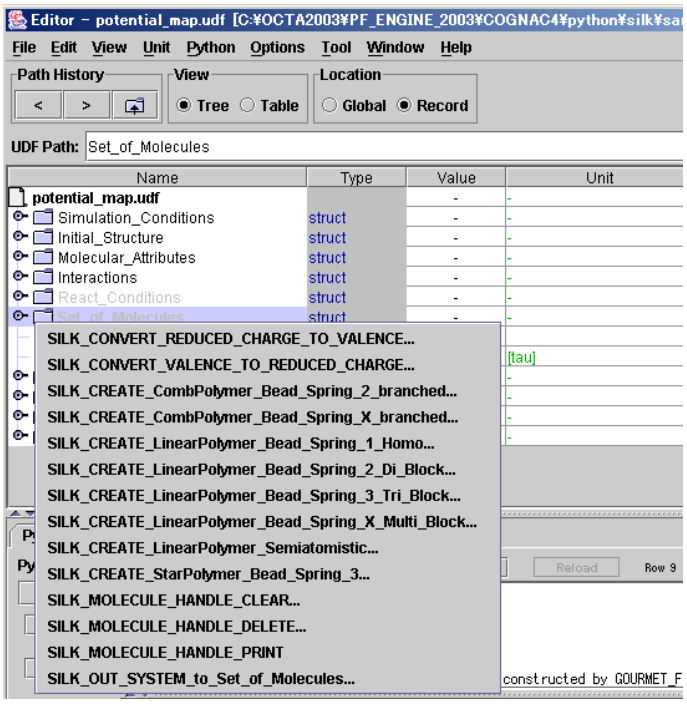


図 6.1: **ACTION** の起動

- **SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block**
  ビーズスプリングモデルの線状ジブロック高分子（例：**A10-B10**）を作成する。

- **SILK_CREATE_LinearPolymer_Bead_Spring_3_Tri_Block**
  ビーズスプリングモデルの線状トリブロック高分子（例：**A10-B10-C10**）を作成する。

- **SILK_CREATE_LinearPolymer_Bead_Spring_X_Multi_Block**
  ビーズスプリングモデルの任意の数の線状マルチブロック高分子（例：**A20-B40-A20-C10**）を作成する。

- **SILK_CREATE_LinearPolymer_Semiatomistic**
  ユナイテッドアトムモデルの線状高分子を作成する。

- **SILK_CREATE_Monomer**
  単原子分子を作成する。

- **SILK_CREATE_StarPolymer_Bead_Spring_3**
  ビーズスプリングモデルの星型（3 腕）高分子を作成する。

作成したい分子を選択し、実行（OK ボタンをクリック）すると、**Python Log** パネルにメッセージが表示される。

SILK_CREATE_LinearPolymer_Bead_Spring_2_Di_Block を実行した場合、以下のようなメッセージが表示される。以下は、**A10-B10** のジブロック分子が作成されたことを意味する。

```
Trying to constructing LINEAR_Block_molecules...
[('A', 10), ('B', 10)]
Section of constructing LINEAR_Block_Polymers. Done.
```

続けて別の分子を作成することができる。

SILK_CREATE_LinearPolymer_Bead_Spring_1_Homo を実行した場合、以下のメッセージが表示される。これは、**A10** のホモ高分子が作成されたことを意味する。

```
Trying to constructing LINEAR_Homo_molecules...
[('A', 10)]
Section of constructing LINEAR_Block_Polymers. Done.
```

もし、作成された分子の種類とそれぞれの本数を確認したい場合は、メニューから
SILK_MOLECULE_HANDLE_PRINT を実行する。以下は実行結果のメッセージである

```
============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. **Set_of_Molecules** への書き込み

   アクションのメニュー **SILK_OUT_SYSTEM_to_Set_of_Molecules** は、作成した分子を **GOURMET** のワークシート（**Set_of_Molecules**）に書き込むコマンドである。以下は実行結果のメッセージである。

   ```
   try_to_write_Set_of_Molecules ...
   item(MOL_NAME,NUM_OF_MOL) [('linear_di', 1), ('linear_homo', 1)]
   ```

   そして、**GOURMET** のワークシート（**Set_of_Molecules**）に、作成した分子が書き込まれる。

4. UDF ファイル (**COGNAC** 入力ファイル) の作成

   GOURMET 画面の左側にはエクスプローラー様のデータ構造が表示されている。このエリアの最上部の、ファイル名が表示されている場所にカーソルを置き、右ボタンクリックするとアクションのメニューがポップアップされ、**SILK_UDF_CREATE** というコマンドが現れる。

   この **SILK_UDF_CREATE** を選択すると、ファイル指定のウィンドウが起動する

   このダイアログの **values** コラムで右ボタンをクリックすると、ファイル選択のウィンドウがポップアップする。

   ディレクトリを選択し、ファイル名を指定すると COGNAC 入力ファイルが作成される。

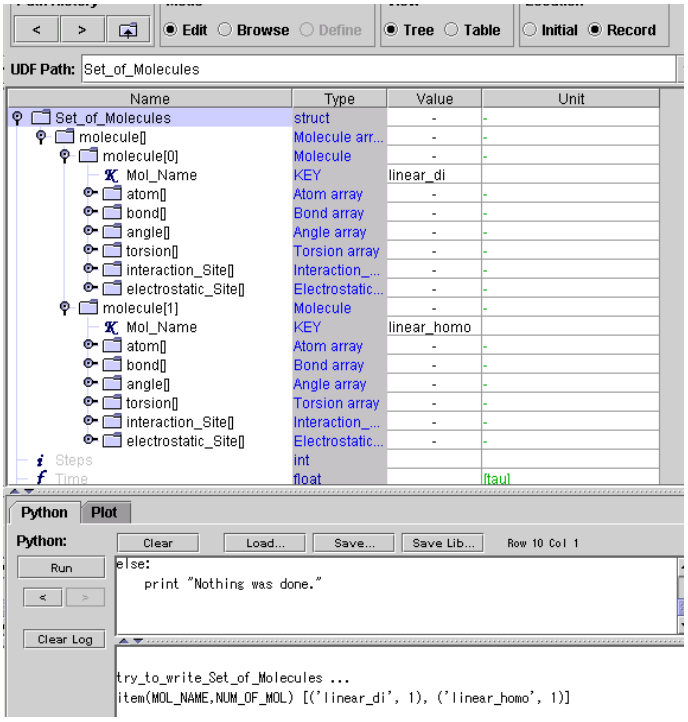


図 6.2: “SILK_OUT_SYSTEM_to_Set_of_Molecules” を実行した結果

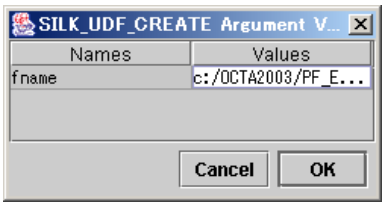


図 6.3: ファイル指定ウィンドウ

6.2　SILK 基本関数を用いた COGNAC 入力ファイル作成の概要

SILK 基本関数を用いた **COGNAC** 入力ファイルの作成手順を以下に示す。この項は、6.1 の項よりもわかり難いが、**SILK** の基本的な機能を知る上で重要である

1. Potential_Map の編集

   SILK を実行する為には、シミュレーションの条件ならびにポテンシャルが記述された **SILK** 用入力 UDF が必要である。**SILK** 用入力 UDF は、**COGNAC** 入力 UDF とほぼ同じ構造を持つ。雛型として “potential_map.udf” が用意されているので、適宜編集して使用する。

2. User editable Python script の Load

   SILK の実体は、複数のモジュールから構成されたパイソンスクリプト (*.py ファイル) 群である。その中のひとつの特定のスクリプト (ファイル名は任意であるが、サンプル例としていくつかの “silk_use_*.py” を用意している) を **GOURMET** 上に Load して使用する。

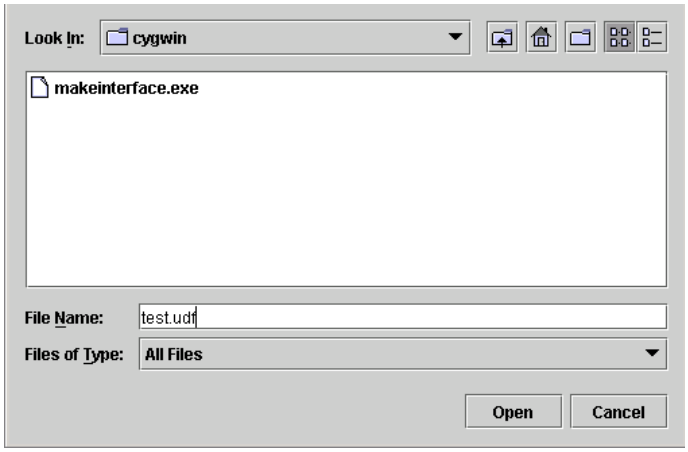


図 6.4: ファイル選択ウィザード

3. User editable Python script の編集

   上に述べた特定のスクリプト（以下、“silk_use.py”）には、**SILK** が作成する **COGNAC** ファイルの出力先や、高分子の作成手順等、使用者が適宜編集する個所がある。編集したスクリプトについては使用者が任意に保存し、再度 Load して使用することが可能である。

4. Python（User editable Python script）の Run

   使用者が編集した”silk_use.py”を **GOURMET** 上で実行することによって、指定されたディレクトリに **COGNAC** 入力ファイルが出力される。**GOURMET** には **Python** 実行結果を表示するウインドウがあり、実行結果が出力される。

6.2.1 SILK 実行の準備（Potential_Map の編集）

SILK を実行する為には、シミュレーションの条件ならびにポテンシャルが記述された **SILK** 用入力 UDF(名前は任意。ここでは” potential_map.udf” と呼ぶ) が必要である。**SILK** 用入力 UDF は、**COGNAC** 入力 UDF とほぼ同じ構造を持っている。使用者は自分の実施したいシミュレーションに合わせて、ポテンシャルのパラメータや温度や圧力等の条件を編集しなければならない。雛型として用意されている “potential_map.udf” を例にとって説明する。

- UDF の Open

  GOURMET を起動し、UDF（ “potential_map.udf”）を Open する。Open 時、画面には Record Number が **0** のデータが表示されている。窓の下のつまみを右へスライドさせ、右下の窓に Record Label である “UA_PE_Kuwajima” が表示されるレコード（Record **2**）であることを確認する。

  起動画面の左側にはエクスプローラー様のデータ構造が表示されている。このデータ構造をクリックすることによって現在の Record のデータを参照（編集）することができる。

- UDF の編集

  使用者は表形式のカラムのデータを編集することができる。

  – **Molecular_Attributes**

UDF Path 名 **Molecular_Attributes** ··· 分子内ポテンシャルに関する各パラメータ。

Molecular_Attributes.Atom_Type[] ··· Atom のタイプに関するデータが収められている。
その中のデータ **Name** は、**SILK** で分子を作成する際に対応付けられるので、対応するように設定する必要がある。

Molecular_Attributes.Bond_Potential[] ··· Bond のタイプに関するデータが収められている。
その中のデータ **Name** は、**SILK** で分子を作成する際に対応付けられるので、対応するように設定する必要がある。

Molecular_Attributes.Angle_Potential[] ··· Angle のタイプに関するデータが収められている。
その中のデータ **Name** は、**SILK** で分子を作成する際に対応付けられるので、対応するように設定する必要がある。

Molecular_Attributes.Torsion_Potential[] ··· Torsion のタイプに関するデータが収められている。
その中のデータ **Name** は、**SILK** で分子を作成する際に対応付けられるので、対応するように設定する必要がある。

Molecular_Attributes.Interaction_Site_Type[] ··· 分子間相互作用や外場を作用させるサイト Interaction Site のタイプに関するデータが収められている。
その中のデータ **Name** は、**SILK** で分子を作成する際に対応付けられるので、対応するように設定する必要がある。

– **Interactions**
UDF Path 名 **Interactions** ··· 分子間ポテンシャルや外場を定義するポテンシャルの各パラメータ。

Interactions.Pair_Interaction[] ··· 分子間相互作用に関するデータが収められている。
その中のデータ **Name** は、**SILK** で作成した分子が持つ Interaction Site のタイプの名前に関連して対応付けられる。

Interactions.External_Interaction[] ··· 外場による作用に関するデータが収められている。
その中のデータ **Site_Name** は、**SILK** で作成した分子が持つ Interaction Site のタイプの名前に関連して対応付けられる。

Interactions.Coulomb_Interaction[] ··· 静電相互作用に関するデータが収められている。
その中のデータ **Name** は、静電相互作用を受けるサイト Electrostatic Site を登録する際に指定した名前に関連して対応付けられる。

– **Unit_Parameter**
UDF Path 名 **Unit_Parameter** ··· 単位換算を行うために必要な、質量、エネルギー、長さの単位あたりの換算値

（**COGNAC** のシミュレーション自身には必要ではないのでブランクでも良い）。

Unit_Parameter のデータについて次に記す。

Mass ··· 質量の単位。
Energy ··· エネルギーの単位。
Length ··· 長さの単位。

これらの単位を決定することにより、温度、圧力等各緒量が計算できる。単位換算のスクリプトととして、“silk_mujigenkun.py” を Load して使用することができる。

– **Initial_Structure**
UDF Path 名 **Initial_Structure**··· 初期構造（各 Atom の座標に関する情報）に関するパラメータ。

– **Simulation_Conditions**
UDF Path 名 **Simulation_Conditions** ··· アルゴリズムの選択等の計算条件に関するパラメータ。

- UDF の Save について

今までに述べた各パラメータについて編集操作を施した場合、編集結果が反映されるように **GOURMET** のメニューから Save をしなければならない。

6.2.2 SILK による高分子トポロジー作成（User editable Python script の編集）

- パイソンスクリプトの Load

GOURMET 上で “silk_use.py” を Load する。Python ウインドウに Load されたスクリプトが表示されていることを確認し、スクリプトの上部を参照、適宜編集する。編集する項目は

– 出力先の指定
– 分子の編集

である。

- Load されたパイソンスクリプトの編集

Load されたスクリプトの上部にユーザーが編集すべきセクション（**SECTION “USER DEFINITION”**）が記述されている。

– 出力先
“USER DEFINITION” 内の **SUBSECTION “outputpath”** について説明する。

```
####### 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="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"
```

self.engine.outDir には出力先のディレクトリを指定する。ディレクトリ階層の区分記述に、スラッシュを用いているが、Windows 環境においても問題は無い。

self.engine.cognacFileName には出力ファイル名（UDF 拡張子：.udf を除く）を指定する。

self.engine.prjName には出力ファイルのプロジェクト名を指定する。

self.engine.fileNumCom には出力ファイルを分割する（トポロジーを記述したファイルとその他の条件を記述したファイルに分割する）か否かのコマンドを記述する。
self.engine.fileNumCom=”ONE_FILE” とした場合、単一のファイルにまとめられる。

– 分子の定義
“USER DEFINITION” 内の **SUBSECTION “system”** について説明する。Load されたスクリプトに下記の記述部があることを確認する。

```
#######  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. 分子の登録

```
name="mol"
```

まず最初に分子（名前）を登録する。**"mol"**は分子に設定される名前である。

```
numMol=10
```

numMol は分子の本数である。

```
self.engine.createMolecule(name)
```

分子を登録する関数である。**self.engine.** は不可欠な記述である。この関数は **name** の値:**"mol"** という名前で **SILK** に対して分子を登録したことを意味する。

2. 登録された分子（分子名:**"mol"**）への Atom の登録

```
self.engine.addAtoms(name, "UA", "UA\_Kuwajima")
```

Atom を登録する関数である。**self.engine.** は不可欠な記述である。
第 1 引数である **name** は、先程の分子名**"mol"**が代入されているので、**"mol"**が入力される。
第 2 引数である**"UA"**は、登録する Atom の名前である。ここで使用者が任意に選べる。
第 3 引数である**"UA_Kuwajima"**は、Atom type の名前である。
UDF Path 名 **Molecular_Attributes.Atom_Type**[] の配列化されたデータの内、**Name** が **"UA_Kuwajima"**であるものが参照される。

次に、**SILK** に対して登録した分子**"mol"**についてシーケンシャルに 4 個の Atom （名前は **"UA"**、Atom type の名前は**"UA_Kuwajima"**）を登録する例を記述する。

```
for i in range(0,4):
  self.engine.addAtoms(name, "UA", "UA_Kuwajima")
```

Python の文法に従い、for ループを用いてシーケンシャルに 4 個の Atom を登録している。

3. 登録された分子（分子名:**"mol"**）への Bond の登録

```
self.engine.addBonds(name,0,1, "BOND_PE_Kuwajima")
```

Bond を登録する関数 **addBonds(...)** を呼び出して Bond を登録する。**self.engine.** は不可欠な記述である。
第 1 引数である **name** は、先程の分子名**"mol"**が代入されているので、**"mol"**が入力される。
第 2 引数である **0** は、登録する Bond を形成する Atom の配列 Index である。
第 3 引数である **1** は、登録する Bond を形成する Atom の配列 Index である。

第 4 引数である**"BOND_PE_Kuwajima"**は、BondType の名前である。
UDF Path 名**"Molecular_Attributes.Bond_Type[]"**の配列化されたデータの内、**"Name"**が**"BOND_PE_Kuwajima"**に合致するものが参照される。
次に、**SILK** に対して登録した分子**"mol"**についてシーケンシャルに 3 個の Bond（Bond type の名前は**"BOND_PE_Kuwajima"**）を登録する例を記述する。

```
for i in range(0,3):
  self.engine.addBonds(name, i, i+1, "BOND_PE_Kuwajima")
```

Python の文法に従い、for ループを用いてシーケンシャルに 3 個の Bond を登録している。

4. 登録された分子（分子名:"mol"）への Angle の登録

```
self.engine.addAngles(name,0,1,2, "ANGLE_PE_Kuwajima")
```

Angle を登録する関数 **addAngles(...)** を呼び出して Angle を登録する。**self.engine.** は不可欠な記述である。
第 1 引数である **name** は、先程の分子名**"mol"**が代入されているので、**"mol"**が入力される。
第 2 引数である **0** は、登録する Angle を形成する Atom の配列 Index である。
第 3 引数である **1** は、登録する Angle を形成する（中央の）Atom の配列 Index である。
第 4 引数である **2** は、登録する Angle を形成する Atom の配列 Index である。
第 5 引数である**"ANGLE_PE_Kuwajima"**は、Angle type の名前である。
UDF Path 名 **Molecular_Attributes.Angle_Type[]** の配列化されたデータの内、**"Name"**が**"ANGLE_PE_Kuwajima"**に合致するものが参照される。
次に、**SILK** に対して登録した分子**"mol"**についてシーケンシャルに 2 個の Angle（Angle type の名前は**"ANGLE_PE_Kuwajima"**）を登録する例を記述する。

```
for i in range(0,2):
  self.engine.addAngles(name, i, i+1, i+2, "ANGLE_PE_Kuwajima")
```

Python の文法に従い、for ループを用いてシーケンシャルに 2 個の Angle を登録している。

5. 登録された分子（分子名:**"mol"**）への Torsion の登録

```
self.engine.addTorsions(name,0,1,2,3, "TORSION_PE_Kuwajima")
```

Torsion を登録する関数 **addTorsions(...)** を呼び出して Torsion を登録する。**self.engine.** は不可欠な記述である。
第 1 引数である **name** は、先程の分子名**"mol"**が代入されているので、**"mol"**が入力される。
第 2 引数である **0** は、登録する Torsion を形成する Atom の配列 Index である。
第 3 引数である **1** は、登録する Torsion を形成する Atom の配列 Index である。
第 4 引数である **2** は、登録する Torsion を形成する Atom の配列 Index である。
第 5 引数である **3** は、登録する Torsion を形成する Atom の配列 Index である。
第 6 引数である**"TORSION_PE_Kuwajima"**は、Torsion type の名前である。

UDF Path 名 **Molecular_Attributes.Torsion_Type**[] の配列化されたデータの内、**Name** が**"TORSION_PE_Kuwajima"**に合致するものが参照される。
次に、**SILK** に対して登録した分子**"mol"**についてシーケンシャルに 1 個の Torsion（Torsion-Type の名前は**"TORSION_PE_Kuwajima"**）を登録する例を記述する。

```
for i in range(0,1):
  self.engine.addTorsions(name, i, i+1, i+2, i+3, "TORSION_PE_Kuwajima")
```

Python の文法に従い、for ループを用いてシーケンシャルに 1 個の Torsion を登録している。

6. 登録された分子（分子名:**"mol"**）への Interaction Site の登録

```
self.engine.addInteractionSites(name, [0], "NB_PE_Kuwajima", "PAIR")
```

分子間相互作用を作用させる Interaction Site を登録する関数 **addInteractionSites(...)** を呼び出して（分子間相互作用に関する）Interaction Site を登録する。
第 1 引数である **name** は、先程の分子名**"mol"**が代入されているので、**"mol"**が入力される。
第 2 引数である **[0]** は、Atom のインデックスの配列を示す。ここで配列は **COGNAC** の特徴である" 複数の Atom から定義される Interaction Site をサポートする" 機能に対応したものである。
第 3 引数である**"NB_PE_Kuwajima"**は Interaction site type の名前である。
UDF Path 名 **Molecular_Attributes.Interaction_Site_Type**[] の配列化されたデータの内、**Name** が**"NB_PE_Kuwajima"**に合致するものが参照される。

7. 登録された分子（分子名:**"mol"**）の内、**COGNAC** 入力ファイルへ出力する分子を設定

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

ここでは登録した分子（名前:**"mol"**）について、何本の分子を出力するかを指定する。**numMol** は分子の本数である。**SILK** に分子を登録、作製しても、この関数 **setSystem(...)** が呼ばれない限り出力はされないので注意。

- パイソンスクリプトの実行

  編集したパイソンスクリプトの内容（出力先等）を確認し、スクリプトを実行する。実行結果についてはログウインドウに表示される。

6.3 機能

6.3.1 出力先

SILK において、出力先は関数 **setOutParam()** 内で指定する。使用者は **setOutParam()** 内を編集することにより出力先等を指定できる。**setOutParam()** は、サンプル "silk_use.py" 等の上部に書かれている（参照のこと）。

self.engine.outDir ··· 出力先のディレクトリ。例えば、

```
self.engine.outDir="C:/OCTA8.3/ENGINES/COGNAC/python/silk/sample"
```

self.engine.cognacFileName ··· ファイル名。
例えば

```
self.engine.cognacFileName="peo"
```

と記述することにより、”peo.udf”が出力ファイル名となる。

self.engine.prjName ··· 出力ファイルのプロジェクト名。
例えば

```
self.engine.prjName="DEVELOP"
```

と記述することにより、**DEVELOP** が出力ファイル（**COGNAC** 入力ファイル）のプロジェクト名となる。

self.engine.fileNumCom ··· 出力ファイルの形態。
例えば

```
self.engine.fileNumCom="ONE\_FILE"
```

とすることにより、“peo.udf” が出力ファイルとなる。

```
self.engine.fileNumCom="TWO\_FILES"
```

とすれば、各種入力条件が記述された “peo.udf” および、分子のトポロジーのみを記述した “peo_str.udf” の二つのファイルが出力される。

6.3.2 分子の登録

```
self.engine.createMolecule(name)
```

第 1 引数（**name**）··· 登録する分子の名前。

6.3.3 atom の登録

```
self.engine.addAtoms(name, AtomName, AtomTypeName)
```

第 1 引数（**name**）··· 登録された分子の名前。
第 2 引数（**AtomName**）··· 登録する Atom の名前。使用者が任意に選ぶことができる。
第 3 引数（**AtomTypeName**）··· 登録する AtomType の名前。

- 第 3 引数について

  UDF Path 名 **Molecular_Attributes.Atom_Type**[] の

  配列化されたデータの **Name** に合致するものが **COGNAC** 実行時に参照される。

6.3.4 atom の各種パラメータの設定

```
self.engine.setAtomParam(molName, seq, paramName, param)
```

第 1 引数（**molName**）··· 登録された分子の名前。
第 2 引数（**seq**）··· パラメータを設定する Atom の配列 Index。
第 3 引数（**paramName**）··· パラメータ名。予約語。
第 4 引数（**param**）··· パラメータの値。

- 第 3 引数、第 4 引数は設定したいパラメータの名前とセットする値の組み合わせ

  "Chirality" ··· **0** あるいは **1**

  "Main_Chain" ··· **0** あるいは **1**

  "Atom_ID" ··· atom ID(任意の整数だが、異なる Atom に同じ ID を用いない方が良い)

6.3.5 bond の登録

```
self.engine.addBonds(name, atom1, atom2, BondTypeName)
```

第 1 引数（**name**）··· 登録された分子の名前。
第 2 引数（**atom1**）··· 登録する Bond を形成する Atom の配列 Index。
第 3 引数（**atom2**）··· 登録する Bond を形成する Atom の配列 Index。
第 4 引数（**BondTypeName**）··· Bond type の名前。

- 第 4 引数について

  UDF Path 名 **Molecular_Attributes.Bond_Type[]** の

  配列化されたデータの **Name** に合致するものが **COGNAC** 実行時に参照される。

6.3.6 angle の登録

```
self.engine.addAngles(name, atom1, atom2, atom3, AngleTypeName)
```

第 1 引数（**name**）··· 登録された分子の名前。
第 2 引数（**atom1**）··· 登録する Angle を形成する Atom の配列 Index。
第 3 引数（**atom2**）··· 登録する Angle を形成する Atom の配列 Index。
第 4 引数（**atom3**）··· 登録する Angle を形成する Atom の配列 Index。
第 5 引数（**AngleTypeName**）··· Angle type の名前。

- 第 5 引数について

  UDF Path 名 **Molecular_Attributes.Angle_Type[]** の

  配列化されたデータの **Name** に合致するものが **COGNAC** 実行時に参照される。

6.3.7　torsion の登録

```
self.engine.addTorsions(name, atom1, atom2, atom3, atom4, TorsionTypeName)
```

第 1 引数（**name**）··· 登録された分子の名前。
第 2 引数（**atom1**）··· 登録する Torsion を形成する Atom の配列 Index。
第 3 引数（**atom2**）··· 登録する Torsion を形成する Atom の配列 Index。
第 4 引数（**atom3**）··· 登録する Torsion を形成する Atom の配列 Index。
第 5 引数（**atom4**）··· 登録する Torsion を形成する Atom の配列 Index。
第 6 引数（**TorsionTypeName**）··· Torsion type の名前。

- 第 6 引数について

  UDF Path 名 **Molecular_Attributes.Torsion_Type**[] の

  配列化されたデータの **Name** に合致するものが **COGNAC** 実行時に参照される。

6.3.8　Interaction Site（分子間相互作用）の登録

```
self.engine.addInteractionSites(name,[atomID], InteractionSiteTypeName, "PAIR")
```

第 1 引数（**name**）··· 登録された分子の名前。
第 2 引数（[**atomID**]）··· Atom の Index の配列。
第 3 引数（**InteractionSiteTypeName**）··· InteractionSiteType の名前。第 4 引数（**”PAIR”**）··· 予約語。

- 第 3 引数について

  UDF Path 名 **Molecular_Attributes.Interaction_Site_Type**[] の

  配列化されたデータの **Name** に合致するものが **COGNAC** 実行時に参照される。

6.3.9　Interaction Site（外場）の登録

```
self.engine.addInteractionSites(name,[atomID], InteractionSiteTypeName, "EXTERNAL")
```

第 1 引数（**name**）··· 登録された分子の名前。
第 2 引数（[**atomID**]）··· Atom の Index の配列。
第 3 引数（**InteractionSiteTypeName**）··· InteractionSiteType の名前。
第 4 引数（**”EXTERNAL”**）··· 予約語。

- 第 3 引数について

  UDF Path 名”**Molecular_Attributes.Interaction_Site_Type**[]”の

  配列化されたデータの **Name** に合致するものが **COGNAC** 実行時に参照される。

6.3.10 Electrostatic Site（静電相互作用）の登録

```
self.engine.addInteractionSites(name,[atomID], ESInteractionName, "COULOMB",ESvalue)
```

第 1 引数（**name**）··· 登録された分子の名前。
第 2 引数（[**atomID**]）··· Atom のシーケンシャル ID の配列を示す。
第 3 引数（**ESInteractionName**）··· ElectroStaticInteraction の名前。
第 4 引数（**"COULOMB"**）··· 予約語。編集してはならない。
第 5 引数（**ESvalue**）··· 双極子能率もしくは電荷。

6.3.11 分子数の指定

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

第 1 引数（**name**）··· 登録された分子の名前。
第 2 引数（**numMol**）··· 分子の本数。

6.4 高分子テンプレート

6.4.1 線状ホモポリマー I（ユナイテッドアトムレベルまで）

図 6.5 に示されるように、ビーズスプリングモデルからユナイテッドアトムレベルまでの線状ホモポリマーを任意本数作製する。

```
self.engine.makeLinPolym(name, numAtom, numMol, atomName, atomTypeName, bondTypeName,
                         angleTypeName, torsionTypeName, interactionSiteTypeName)
```

第 1 引数（**name**）··· 分子の名前。
第 2 引数（**numAtom**）··· 1 分子当りの Atom の個数。
第 3 引数（**numMol**）··· 分子の本数。
第 4 引数（**atomName**）··· Atom の名前。
第 5 引数（**atomTypeName**）··· Atom type の名前。
第 6 引数（**bondTypeName**）··· Bond potential の名前。
第 7 引数（**angleTypeName**）··· Angle potential の名前。
第 8 引数（**torsionTypeName**）··· Torsion potential の名前。
第 9 引数（**interactionSiteTypeName**）··· Interaction site type の名前。

- 第 7 引数について
  名前を指定しなかった場合（""とした場合）、Angle はセットされない。
- 第 8 引数について
  名前を指定しなかった場合（""とした場合）、Torsion はセットされない。
- 第 9 引数について
  名前を指定しなかった場合（""とした場合）、Interaction Site はセットされない。

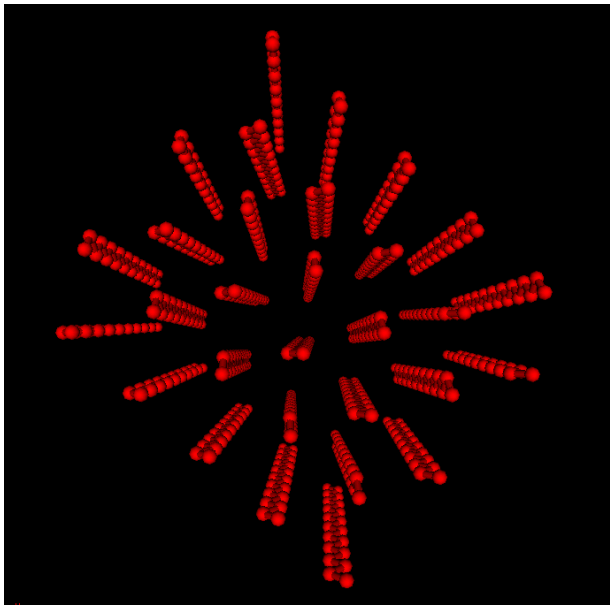


図 6.5: **SILK** で作成したアルカン

6.4.2　線状ホモポリマー II（分子量分布を持つ場合）

ビーズスプリングモデルからユナイテッドアトムレベルまでの線状ホモポリマーを分子量分布を持たせて任意本数作製する。

分布については、**FORK**（レオロジーシミュレータ **PASTA** のプリプロセッサ）の出力ファイルから読み込んでトポロジーを生成する。**FORK** の詳細に関しては「PASTA ユーザーズマニュアル」を参照のこと

```
self.engine.makeLinPolymMWD(mwdFileName, baseName, numBeadInZ, chainNumScale,
                            atomName, atomTypeName, bondTypeName, angleTypeName,
                            torsionTypeName, interactionSiteTypeName)
```

第 1 引数（**mwdFileName**）··· ファイルの名前。
第 2 引数（**baseName**）··· 分子の名前。
第 3 引数（**numBeadInZ**）··· Z（絡み合い点間分子量）当りの Atom の個数。
第 4 引数（**chainNumScale**）··· 各分子量の分子の本数と作成される分子のスケール比。
第 5 引数（**atomName**）··· Atom の名前。
第 6 引数（**atomTypeName**）··· Atom type の名前。
第 7 引数（**bondTypeName**）··· Bond potential の名前。
第 8 引数（**angleTypeName**）··· Angle potential の名前。
第 9 引数（**torsionTypeName**）··· Torsion potential の名前。
第 10 引数（**interactionSiteTypeName**）··· Interaction site type の名前。

- 第 2 引数について

  例えば”**mol**”とした場合、各分子の名前は”**mol_1**”,”**mol_2**”…と名づけられる。

- 第 8 引数について

  名前を指定しなかった場合（””とした場合）、Angle はセットされない。

- 第 9 引数について

  名前を指定しなかった場合（””とした場合）、Torsion はセットされない。

- 第 10 引数について

  名前を指定しなかった場合（””とした場合）、Interaction Site はセットされない。

6.4.3 線状マルチブロックポリマー（ビーズスプリングモデル）

図 6.6 に示されるような、ビーズスプリングモデルでのマルチブロックポリマーを任意本数作製する。

```
self.engine.makeBeadSpringPolym(name, numMol, "LINEAR", sequence,
                                atomType, bondType, interactionSiteType)
```

第 1 引数（**name**）··· 分子の名前。
第 2 引数（**numMol**）··· 分子の本数。
第 3 引数（**”LINEAR”**）··· 予約語。
第 4 引数（**sequence**）··· ブロックシーケンス。
第 5 引数（**atomType**）··· sequence で記述された Atom と Atom type からなる辞書。
第 6 引数（**bondType**）··· sequence で記述された Atom 間の Bond と Bond potential からなる辞書。
第 7 引数（**interactionSiteType**）··· sequence で記述された Atom とそこから定義される Interaction site type からなる辞書。

- 第 4 引数について

  A20-B40-A20 のトリブロックポリマーを作製する場合、次のように記述する。

  [(”A”,20),(”B”,40),(”A”,20)] （**”A”**、**”B”**という名前の Atom が登録される）

- 第 5 引数について

  例えば **{”A”:”atom1”, ”B”:”atom2”}** と記述すると

  Atom **”A”**のタイプとして**”atom1”**が、Atom **”B”** のタイプとして**”atom2”**が登録される。

- 第 6 引数について

  例えば **{”A_A”:”bond1”, ”A_B”:”bond3”, ”B_B”:”bond2”}** と記述すると

  Atom **”A”**と Atom **”A”**が作る Bond の potential name として**”bond1”**が、

  Atom **”A”**と Atom **”B”**が作る Bond の potential name として**”bond3”**が、

  Atom **”B”**と Atom **”B”**が作る Bond の potential name として**”bond2”**が登録される。

- 第 7 引数について

  例えば **{”A”:”siteType1”, ”B”:”siteType2”}** と記述した場合

  Atom **”A”**によって定義される Interaction site のタイプとして**”siteType1”**が、

  Atom **”B”**によって定義される Interaction site のタイプとして**”siteType2”**が登録される。

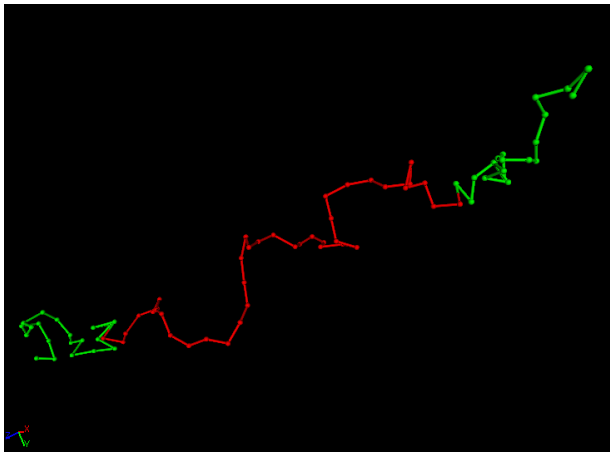


図 6.6: **SILK** で作成したトリブロックポリマー

6.4.4　櫛形ポリマー（ビーズスプリングモデル）

図 6.7 に示されるように、ビーズスプリングモデルでの櫛形ポリマーを任意本数作製する。また図 6.8 に示されるように星形ポリマーの作製も可能である。

```
self.engine.makeBeadSpringPolym(name, numMol, "COMB", sequence,
                                atomType, bondType, interactionSiteType)
```

第 1 引数（**name**）··· 分子の名前。
第 2 引数（**numMol**）··· 分子の本数。
第 3 引数（**”COMB”**）··· 予約語。
第 4 引数（**sequence**）··· ブロックシーケンス。
第 5 引数（**atomType**）··· sequence で記述された Atom と Atom type からなる辞書。
第 6 引数（**bondType**）··· sequence で記述された Atom 間の Bond と Bond type からなる辞書。
第 7 引数（**interactionSiteType**）··· sequence で記述された Atom と Atom から定義される Interaction site type からなる辞書。

- 第 4 引数について

  A20(B20)-A20(B20)-A20 の櫛形ポリマー（B20 が分岐鎖である）を作製する場合、次のように記述する。

  [(”A”,20),(”B”,20),(”A”,20),(”B”,20),(”A”,20)]（**”A”**、**”B”** という名前の Atom が登録される）

  また、**[(”A”,20),(”B”,20),(”A”,0),(”B”,20),(”A”,20)]** と記述した場合

  A60 の主鎖に、B20 の鎖が 2 本分岐した構造（星形ポリマー）となる。

  図 6.8 に、**[(”A”,20),(”B”,20),(”A”,0),(”B”,20),(”A”,20)]** で作成された分子を示す。

- 第 5 引数について

  例えば **{”A”:”atom1”, ”B”:”atom2”}** と記述すると

  Atom **”A”**のタイプとして**”atom1”**が、Atom **”B”** のタイプとして**”atom2”**が登録される。

- 第 6 引数について

  例えば **{"A_A":"bond1", "A_B":"bond3", "B_B":"bond2"}** と記述すると

  Atom **"A"**と Atom **"A"**が作る Bond の potential name として**"bond1"**が、

  Atom **"A"**と Atom **"B"**が作る Bond の potential name として**"bond3"**が、

  Atom **"B"**と Atom **"B"**が作る Bond の potential name として**"bond2"**が登録される。

- 第 7 引数について

  例えば **{"A":"siteType1", "B":"siteType2"}** と記述した場合

  Atom **"A"**によって定義される Interaction site のタイプとして**"siteType1"**が、

  Atom **"B"**によって定義される Interaction site のタイプとして**"siteType2"**が登録される。

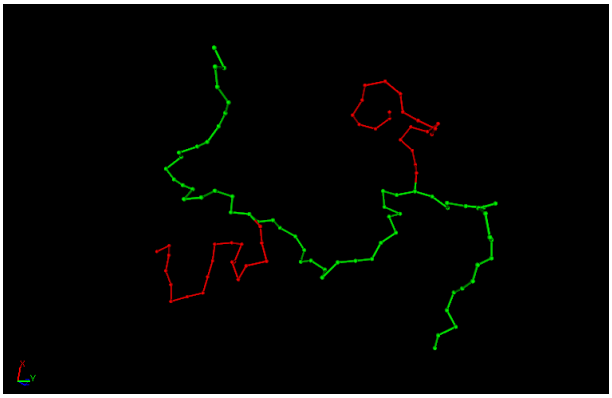


図 6.7: **SILK** で作成した楕形ポリマー（テンプレート関数 **self.engine.makeBeadSpringPolym(...)**）によって作成、シーケンスは [("A",20),("B",20),("A",20),("B",20),("A",20)] ）

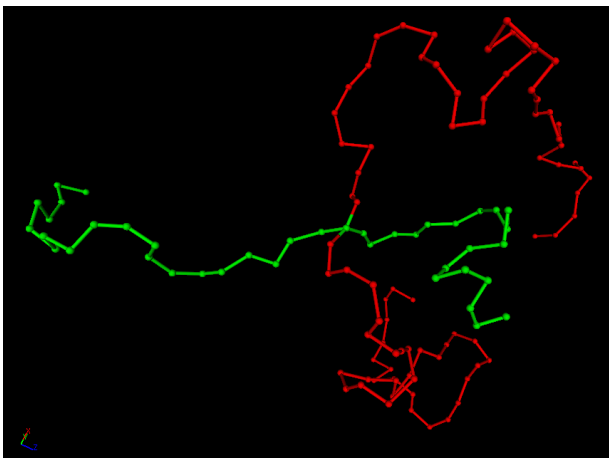


図 6.8: **SILK** で作成した星形ポリマー（テンプレート関数 **self.engine.makeBeadSpringPolym(...)**）によって作成、シーケンスは [("A",20),("B",20),("A",0),("B",20),("A",20)]）

第7章　出力結果解析用ツール

UDF データの解析、フォーマット変換、**GOURMET** におけるデータのグラフィックス表示等を行うため、Python script を使用する。また、一部の解析に関しては、これらの Python script を利用するための **Action** コマンドも同時に用意されている。**Action** によりユーザーは **GOURMET** のグラフィカルユーザーインターフェースにより、Python script を起動することが出来る。

本章では **COGNAC** 関連で用意された Python script および **Action** コマンドに関して解説する。

なお、以下に紹介するスクリプトファイル、**Action** コマンドファイルは、各々“python/analysis” および “action” ディレクトリ以下に用意されている。

7.1　スクリプト一覧

COGNAC の計算結果を、表示／解析するために以下の Python script/Library が用意されている。

- “CognacShowLib.py” ··· 分子構造の表示
- “CognacBasicAnalysis.py” ··· 座標データ、距離等基礎的な諸量の計算
- “CognacGeometryAnalysis.py” ··· 動径分布、密度分布等単一レコードデータの解析
- “CognacTrajectoryAnalysis.py” ··· 平均自乗変位等レコードをまたがるデータの解析
- “CognacFileConvert.py” ··· UDF ファイルの固定形式への変換
- “CognacUtiltity.dll/so” ··· 境界条件による座標変換、自己相関関数等を高速に行うためのライブラリ。入出力は UDF に依存しない

なお、**CognacBasicAnalysis** は **UDFManager** のサブクラスとして設計されているので、以下に解説するメソッド以外に **get**, **jump** 等 **UDFManager** のメソッドを用いることができる。**UDFManager** の使用法に関しては「GOURMET Python スクリプトリファレンスマニュアル」参照のこと。

また、**CognacGeometryAnalysis, CognacTrajectoryAnalysis, CognacFileCovert** は **CognacBasicAnalysis** のサブクラスであり、**CognacBasicAnalysis** のメソッドも用いることができる。

7.2　セットアップ

python ディレクトリを環境変数 **PYTHONPATH** に追加する。またはすべての script/library を Python path の通っているディレクトリにおく。

GOURMET およびシミュレーションプログラム群の通常のインストールに従えば自動的に設定が行われている。

7.3　利用方法

GOURMET あるいは通常の Python command prompt、スクリプトファイルにおいて import して利用できる。ただし、“CognacShowLib.py” は **Viewer** window に分子構造等を表示させるためのスクリプトであるので **GOURMET** python command でのみ利用できる。

7.4　クラスおよびメソッド解説

7.4.1　CognacShowLib.py

クラス

CognacShow(_udf_)
コンストラクタの引数、_**udf**_は表示する UDF object name。**GOURMET** python command で利用する場合は、固定でよい。

メソッド

- **all**(*vtype,bc,color,drawrange*)
  分子集合体全体の表示
  - *vtype* ··· 描画タイプ
    'line'(default)/'ball-stick'/'rod'/'volume'
  - *bc* ··· 描画時の境界条件の設定。
    'mol' あるいは **'on'** の場合、分子の重心がユニットセル内に位置するように境界条件を作用させて表示する
    'atom' の場合、各原子がすべてユニットセル内に位置するように境界条件を作用させて表示する
    'off'(default) の場合、境界条件を考慮せず、Atom の座標値で表示する
  - *color* ··· 描画時の色指定
    'atom'(default) の場合、Atom type で色分けする
    'bond' の場合、Bond type で色分けする
    'mol' の場合、分子毎に色分けする。ただし OCTA がデフォルトで持つ色の設定には限りがあるので、多数の分子を表示するときは、同じ色が繰り返し使われる
    'molname' の場合、Molecular name で色分けする
  - *drawrange* ··· 描画時のユニットセル表示範囲
    基本のユニットセルに含まれる分子に加えて、周囲に存在するイメージも表示することが出来る。表示する範囲はリスト形式で与え、以下の順で範囲を指定する。
    [[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
    amin,amax,bmin... には各々a,b,c 軸方向に表示するセルの範囲を整数で指定する。例えば
    [[-1,1],[-1,1],[-1,1]] と指定した場合、a,b,c 軸すべて、基本セル内の分子と、-1 および 1 シフトしたイメージ分子が表示される。

- **molecule**(*target,vtype,bc,attrid,shift*)
  任意の分子の表示
  - *target* ··· molecule の Index (int)、**Mol_Name** あるいは UDF location を指定する。
    Mol_Name を指定した場合、同じ **Mol_Name** を持つすべての分子が表示される。

- *vtype* ··· 描画タイプ
  'line'(default)/'ball-stick'/'rod'/'volume'
- *bc* ··· 描画時の境界条件の設定。
  'mol' あるいは **'on'** の場合、分子の重心がユニットセル内に位置するように境界条件を作用させて表示する
  'atom' の場合、各原子がすべてユニットセル内に位置するように境界条件を作用させて表示する
  'off'(default) の場合、境界条件を考慮せず、Atom の座標値で表示する
- *attrid* ··· 表示色の設定
  ID/'atom'(default)/'bond'
  ID を与えた場合、ShapeDrawingAttr file における設定により表示する。
  ShapeDrawingAttr file に関する詳細は「GOURMET Python スクリプト　リファレンス　マニュアル」参照
- *shift* ··· 表示位置のシフト量
  オリジナルのユニットセルに対応した位置から、シフトさせて描画する際のシフト量。[a,b,c] セル軸に対応する整数のベクトルで与える。

- **atom(***target,vtype,bc,attrid***)**
  任意の atom の表示
  - *target* ··· atom の Index **list[molIndex,atomIndex]/[atomIndex]**、**Atom_Type_Name**(string)、または UDF location を指定する
    Index list の場合、List の長さが 2 の場合、指定した **molIndex** の指定した **atomIndex** が表示される。1 の場合、すべての分子の指定された **atomIndex** を持つ atom が表示される。
    Atom_Type_Name で指定した場合は、指定した **Atom_Type_Name** の atom がすべて表示される。
    UDF location は単一の atom の場合のみ有効
  - *vtype* ··· 描画タイプ
    'ball'(default)/'point'
  - *bc* ··· 描画時の境界条件の設定。
    'atom' あるいは **'on'** の場合、指定した Atom がユニットセル内に位置するように境界条件を作用させて表示する
    'mol' の場合、指定した Atom を含む分子の重心がユニットセル内に位置するように境界条件を作用させて表示する
    'off'(default) の場合、境界条件を考慮せず、Atom の座標値で表示する
  - *attrid* ··· 表示色の設定。**'atom'(default)/ID**
    'atom' の場合、**Atom_Type** に応じて表示色が設定される。ID を与えた場合、ShapeDrawingAttr file における設定により表示する

- **bond(***target,vtype,bc,attrid***)**
  任意の bond の表示
  - *target* ··· bond の Index list [**molIndex,bondIndex**]/[**bondIndex**]、**Bond_Potential_Name**、または UDF location を指定する
    Index list の場合、List の長さが 2 の場合、指定した **molIndex** の指定した **bondIndex** が表示さ

れる。1 の場合、すべての分子の指定された **bondIndex** を持つ bond が表示される。
Bond_Potential_Name で指定した場合は、指定した **Bond_Potential_Name** の bond がすべて表示される。
UDF location は単一の bond の場合のみ有効

- *vtype* ··· 描画タイプ
  'line'(default)/'rod'/'stick'
- *bc* ··· 描画時の境界条件の設定。
  'bond' あるいは **'on'** の場合、指定した Bond がユニットセル内に位置するように境界条件を作用させて表示する
  'mol' の場合、指定した Bond を含む分子の重心がユニットセル内に位置するように境界条件を作用させて表示する
  'off'(default) の場合、境界条件を考慮せず、Bond の座標値で表示する

- *attrid* ··· 表示色の設定**'atom'/'bond'(default)/ID**
  'atom'/'bond' の場合、各々**Atom_Type,Bond_Potential** に応じて表示色が設定される。ID を与えた場合、ShapeDrawingAttr file における設定により表示する

- **volume(***index, attrid***)**
  任意の interaction Site の表示。1 atom および 2 atoms から定義される Site を表示することが可能。One atom site の場合は球、Two atom site の場合は楕円体で表示する。

  - *index* ··· interaction site の Index list [**molIndex,siteIndex**]/[**siteIndex**]
    List の長さが 2 の場合、指定した **molIndex** の指定した **siteIndex** が表示される
    1 の場合、すべての分子の指定された **siteIndex** を持つ interaction site が表示される
  - *attrid* ··· 表示色を設定する ID (int)

- **cell(***color,drawrange***)** ··· ユニットセルの表示
  ユニットセルが line で表示される。

  - *color(int)* ··· ShapeDrawingAttr file の設定により色が選択される。デフォルトは **0**（白）
  - *drawrange* ··· ユニットセル表示範囲
    基本のユニットセルに加えて、周囲に存在するイメージも表示することが出来る。表示する範囲はリスト形式で与え、以下の順で範囲を指定する。
    [[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
    amin,amax,bmin... には各々a,b,c 軸方向に表示するセルの範囲を整数で指定する。例えば
    [[-1,1],[-1,1],[-1,1]] と指定した場合、a,b,c 軸すべて、基本セルと、-1 および 1 シフトしたユニットセルが表示される。

- **boundary(***color,drawrange***)** ··· 境界条件の表示
  周期境界あるいは反射境界条件が設定されていない面を表示する。

  - *color(int)* ··· ShapeDrawingAttr file の設定により色が選択される。デフォルトは **0**（白）
  - *drawrange* ··· 描画時の表示範囲
    基本のユニットセルに含まれる境界条件に加えて、周囲に存在するイメージも表示することが出来

る。表示する範囲はリスト形式で与え、以下の順で範囲を指定する。
[[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
amin,amax,bmin... には各々a,b,c 軸方向に表示するセルの範囲を整数で指定する。例えば
[[-1,1],[-1,1],[-1,1]] と指定した場合、a,b,c 軸すべて、基本セル内と、-1 および 1 シフトしたイメージセルに対応する境界条件が表示される。

- **densityField**(*griddensity,nmesh,atom_name, frame_attr,levelvalue,surf_color,clist*) ··· 密度分布の表示
Atom の分布から計算される密度分布に基づいてカラーコンター、等値面を表示する。

  - *griddensity* ··· UDF 中の Grid density のデータを用いるかどうかのフラグ。**'yes'/'no'(default)**
'yes' の場合は、UDF に Grid density のデータが含まれている必要がある。**'no'** の場合は、Structure のデータより密度分布を計算する。
  - *nmesh* ··· 密度分布を計算する一辺あたりの格子点の数。デフォルトは **16**
griddensity=**'yes'** の場合は、Grid density に出力されている Mesh データを用いるので、この指定は無視される。
  - *atom_name* ··· 密度を表示する Atom name の指定。
Atom name のリストとして与える。
  - *frame_attr* ··· フレームを表示する色の属性の指定
  - *levelvalue* ··· 等値面を表示する場合、2 つの値をリストとて与える。2 つの値の中間値が面上の値として等値面が表示される
  - *surf_color* ··· 等値面の色の指定。[R,G,B, 透過度] のリストで指定。デフォルトは **[1,1,0,1]**（黄色）

7.4.2 CognacBasicAnalysis.py

クラス

CognacBasicAnalysis(*udffile,record*)

- *udffile* ··· オープンする UDF file name
- *record* ··· オープンする Record No.

メソッド

- **position**(*atom*) ··· Atom の座標を求める
  - *atom* ··· Atom Index list [**molIndex,atomIndex**]

  return ··· **position (x,y,z)**

- **velocity**(*atom*) ··· Atom の速度を求める
  - *atom* ··· Atom Index list [**molIndex,atomIndex**]

  return ··· **velocity (x,y,z)**

- **force**(*atom*) ··· Atom に作用する力求める

  - *atom* ··· Atom Index list [**molIndex,atomIndex**]

  return ··· **force (x,y,z)**

- **vector(***atom1,atom2***)** ··· 2 原子間の差ベクトルを求める
  - *atom1/atom2* ··· Atom Index lists [**molIndex,atomIndex**]

  return ··· **vector (x2-x1,y2-y1,z2-z1)**

- **distance(***arg1,arg2***)** ··· 2 原子間の距離
  - Usage 1: 同じ **Potential_Name** を持つ Bond の長さを求める
    - *arg1* ··· bond が持つ **Potential_Name**

    return ··· **List** of bond length

  - Usage 2: 同じ **Mol_Name** を持つ分子の、Bond index で特定する Bond の長さを求める
    - *arg1* ··· **Mol_Name**
    - *arg2* ··· Index of bond

    return ··· **List** of bond length

  - Usage 3: Molecular Index/Bond Index で指定された特定の Bond の長さを求める
    - *arg1* ··· **List** of bond index [**molIndex, bondIndex**]

    return ··· Bond length

  - Usage 4: 特定の 2 原子間の距離を求める。2 原子間に結合が定義されていなくてもよい
    - *arg1* ··· Index list of atom1 [**molIndex,atomIndex**]
    - *arg2* ··· Index list of atom2 [**molIndex,atomIndex**]

    return ··· Distance

- **angle(***arg1,arg2,arg3***)** ··· 3 原子のなす角度
  - Usage 1: 同じ **Potential_Name** を持つ Angle の角度を求める
    - *arg1* ··· angle が持つ **Potential_Name**

    return ··· **List** of angle

  - Usage 2: 同じ **Mol_Name** を持つ分子の、Angle index で特定する Angle の角度を求める
    - *arg1* ··· **Mol_Name**
    - *arg2* ··· Index of Angle

    return ··· **List** of angle

  - Usage 3: Molecular Index/Angle Index で指定された特定の Angle の角度を求める

* *arg1* ··· **List** of angle index [**molIndex,angleIndex**]

return ··· Angle

– Usage 4: 特定の 3 原子のなす角度を求める

* *arg1* ··· Index list of atom1 [**molIndex,atomIndex**]
* *arg2* ··· Index list of atom2 [**molIndex,atomIndex**]
* *arg3* ··· Index list of atom3 [**molIndex,atomIndex**]

return ··· Angle

- **torsion(***arg1,arg2,arg3,arg4***)** ··· 4 原子のなす二面角

– Usage 1: 同じ **Potential_Name** を持つ Torsion の二面角を求める

* *arg1* ··· torsion の持つ **Potential_Name**

return ··· **List** of torsion angle

– Usage 2: 同じ Mol_Name を持つ分子の、Torsion index で特定する Torsion の二面角を求める

* *arg1* ··· **Mol_Name**
* *arg2* ··· Index of Torsion

return ··· **List** of torsion angle

– Usage 3: Molecular Index/Torsion Index で指定された特定の Torsion の二面角を求める

* *arg1* ··· **List** of torsion index [**molIndex, torsionIndex**]

return ··· Torsion angle

– Usage 4: 特定の 4 原子のなす二面角を求める

* *arg1* ··· Index list of atom1 [**molIndex,atomIndex**]
* *arg2* ··· Index list of atom2 [**molIndex,atomIndex**]
* *arg3* ··· Index list of atom3 [**molIndex,atomIndex**]
* *arg4* ··· Index list of atom4 [**molIndex,atomIndex**]

return ··· Torsion angle

- **R2(***arg1***)** ··· 分子鎖の末端間距離の自乗 R^2 を求める

– Usage 1: 指定した分子の R^2 を求める

* *arg1* ··· Index of molecule

return ··· $(R^2, (R_x^2, R_y^2, R_z^2))$

– Usage 2: **Mol_Name** で指定した分子の R^2 のリストを求める

* *arg1* ··· **Mol_Name**

return $\cdots$ list of $(R^2, (R_x^2, R_y^2, R_z^2))$

- **Rg2(***arg1***)** $\cdots$ 分子鎖の回転半径の自乗 Rg^2 を求める
  - Usage 1: 指定した分子の Rg^2 を求める
    - *arg1* $\cdots$ Index of molecule

    return $\cdots$ $(Rg^2, (Rg_x^2, Rg_y^2, Rg_z^2))$
  - Usage 2: **Mol_Name** で指定した分子の Rg^2 のリストを求める
    - *arg1* $\cdots$ **Mol_Name**

    return $\cdots$ List of $(Rg^2, (Rg_x^2, Rg_y^2, Rg_z^2))$

- **centerofmass(***molIndex, atomName***)** $\cdots$ 分子鎖あるいは分子中の部分鎖の重心を求める
  - *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 $\cdots$ Center of mass $\boldsymbol{r_{com}}$

- **system_centerofmass()** $\cdots$ 系全体の重心を求める

  return $\cdots$ Center of mass $\boldsymbol{r_{com}}$

- **Xp(***molIndex, p***)** $\cdots$ 分子鎖の Normal coordinate を求める
  - *molIndex* $\cdots$ Index of molecule
  - p $\cdots$ p-th mode (default = 1)

  return $\cdots$ (Xp_x, Xp_y, Xp_z)

- **inertiaAxis(***molIndex, atomName***)** $\cdots$ 分子鎖あるいは分子中の部分鎖の慣性主軸を求める
  戻り値の inertia axis list はモーメント **m** の小さい順にソートされている
  - *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 $\cdots$ Inertia moments **m** and inertia axes **v**
  $([m1, m2, m3], [[v1_x, v1_y, v1_z], [v2_x, v2_y, v2_z], [v3_x, v3_y, v3_z]])$

7.4.3 CognacGeometryAnalysis.py

クラス

CognacGeometryAnalysis(*udffile,record***)**

- *udffile* ··· オープンする UDF file name
- *record* ··· オープンする Record No.

メソッド

- **gr(***atomnames,width,max,type,minangle,maxangle***)** ··· 動径分布関数を求める
  - *atomnames* ··· **List** of **Atom_Name**。**List** の長さが 1 の時は自己相関、2 の時は相互相関を求める
  - *width* ··· ヒストグラム 1 切片のサイズ (default=0.1)
  - *max* ··· ヒストグラムの最大距離 (default=最小セル長の 1/2、ただし staggered reflective boundary の場合は異なる)
  - *type* ··· 動径分布を計算する対象の指定**'all'/'inter'/'intra'**
    'all の場合、**Atom_Name** で指定するすべてのペアの動径分布を計算する。
    'inter' の場合、分子間のペアのみ、
    'intra' の場合、分子内のペアのみの動径分布を計算する
  - *minangle* ··· 扇状平均を取る最小角度
  - *maxangle* ··· 扇状平均を取る最大角度

  return ··· **List** of [**distance, number density, (number density of sector x,y,z)**]

- **gr_mol(***molnames,width,max,minangle,maxangle***)** ··· 分子の重心の動径分布関数を求める
  - *molnames* ··· **List** of **Mol_Name**。**List** の長さが 1 の時は自己相関、2 の時は相互相関を求める
  - *width* ··· ヒストグラム 1 切片のサイズ (default=0.1)
  - *max* ··· ヒストグラムの最大距離 (default=最小セル長の 1/2、ただし staggered reflective boundary の場合は異なる)
  - *minangle* ··· 扇状平均を取る最小角度
  - *maxangle* ··· 扇状平均を取る最大角度

  return ··· **List** of [**distance, number density**]

- **profile1D(***atomnames,direction,property,bin***)** ··· 原子座標の 1 次元方向の密度分布を求める
  - *atomnames* ··· **List** of **Atom_Name**。複数の **Atom_Name** を指定することが可能
  - *direction* ··· 密度分布を計算する軸**'X'/'Y'/'Z'(default)**
  - *property* ··· 解析するプロパティ**'density'(default)/'velocity'**
  - *bin* ··· ヒストグラム切片の数 (default=int(cell size))

return ··· **List** of [**position,** [$\phi 1, \phi 2$**...**]]

- **profile1Dmol(***molnames,direction,property,bin***)** ··· 分子重心位置の 1 次元方向の密度分布を求める
  - *molnames* ··· **List** of **Atom_Name**。複数の **Mol_Name** を指定することが可能
  - *direction* ··· 密度分布を計算する軸**'X'/'Y'/'Z'(default)**
  - *property* ··· 解析するプロパティ**'density'(default)/'velocity'**
  - *bin* ··· ヒストグラム切片の数 (default=int(cell size))

  return ··· **List** of [**position,** [$\phi 1, \phi 2$**...**]]

- **orientationOrderParameter(***molname,orientVec,refVec***)** ··· 配向秩序パラメータを求める
  Mol_Name を指定して、該当する分子の配向度を計算する
  - *molname* ··· **Mol_Name**
  - *orientVec* ··· 配向度を求める 2 原子間のベクトル **r** [**atomIndex1,atomIndex2**] あるいは慣性主軸のインデックス **(0,1 or 2)**
    orientVec=**None(default)** の場合、慣性モーメントが最小となる慣性主軸 (**Index = 0**) が使用される
  - *refVec* ··· 配向度を求める参照となるベクトル $\mathbf{r_0}$=[x,y,z]
    refVec=**None(default)** の場合、*orientVec* で指定するベクトルの平均値 $\langle \mathbf{r} \rangle$ が参照ベクトルとなる

  return ··· List of order parameter given by $(3\mathbf{u} \cdot \mathbf{u_0}^2 - 1)/2$
  $\mathbf{u}, \mathbf{u_0}$:normalized vector of $\mathbf{r}, \mathbf{r_0}$

- **orientationOrderParameterBond(***refVec***)** ··· Bond の配向秩序パラメータを求める
  Set_of_Molecules に含まれるすべての **Potential_Name** についての配向度を計算する
  - *refVec* ··· 配向度を求める参照となるベクトル $\mathbf{r_0}$=[x,y,z]
    refVec=**None(default)** の場合、おなじ **Potential_Name** を持つ全ての bond ベクトルの平均値 $\langle \mathbf{r} \rangle$ が参照ベクトルとなる

  return ··· List of "bond potential name and list of order parameter given by $(3\mathbf{u} \cdot \mathbf{u_0}^2 - 1)/2$
  $\mathbf{u}, \mathbf{u_0}$:normalized vector of $\mathbf{r}, \mathbf{r_0}$ "

- **normalizeOn()/normalizeOff()** ··· gr, density の出力を規格化するかどうかのトグルスイッチ。On の場合、システムのトータル原子の数密度で規格化される

7.4.4 CognacTrajectoryAnalysis.py

クラス

CognacTrajectoryAnalysis(*udffile***)**

- *udffile* ··· オープンする UDF file name

メソッド

- **molMsd(***molname, start_record, end_record, cancel_trans***)** ··· 分子鎖重心の平均自乗変位 MSD を求める
  - *molname* ··· MSD を計算する分子名 **Mol_Name**
  - *start_record* ··· サンプリングの開始レコード No.
  - *end_record* ··· サンプリングの終了レコード No.
  - *cancel_trans* ··· True なら系全体の重心の並進の寄与を補正する (default = False)

  return ··· **List** of $(Time, MSD, (MSD_x, MSD_y, MSD_z))$

- **atomMsd(***molname, atomIndex, start_record, end_record, cancel_trans***)** ··· 指定した Atom の平均自乗変位 MSD を求める
  Mol_Name で指定する分子の、Atom Index で指定する Atom の MSD を求める
  - *molname* ··· MSD を計算する原子を含む分子名 **Mol_Name**
  - *atomIndex* ··· MSD を計算する原子の Index のリスト。例）[0,1,2,3,4]
  - *start_record* ··· サンプリングの開始レコード No.
  - *end_record* ··· サンプリングの終了レコード No.
  - *cancel_trans* ··· True なら系全体の重心の並進の寄与を補正する (defult = False)

  return ··· **List** of $(Time, MSD, (MSD_x, MSD_y, MSD_z))$

- **normalCoodinate(***molname,p,start_record,end_record***)** ··· Normal coordinate の自己相関関数 Cp を求める
  Mol_Name で指定する分子の Cp の平均値を求める
  - *molname* ··· Cp を計算する **Mol_Name**
  - *p* ··· p-th mode (default = 1)
  - *start_record* ··· サンプリングの開始レコード No.
  - *end_record* ··· サンプリングの終了レコード No.

  return ··· **List** of $(Time, Cp, (Cp_x, Cp_y, Cp_z))$

- **vectorAutoCorrelation(***molname,atomIndex,start_record,end_record***)** ··· 指定した原子間ベクトルの自己相関関数を求める
  - *molname* ··· 自己相関関数を計算する原子を含む分子名 **Mol_Name**
  - *atomIndex* ··· ベクトルを定義する 2 つの原子の Index のリスト。例）[0,10]
  - *start_record* ··· サンプリングの開始レコード No.
  - *end_record* ··· サンプリングの終了レコード No.

  return ··· **List** of $(Time, Cvec, (Cvec_x, Cvec_y, Cvec_z))$

- **atomVelocityAutoCorrelation**(*atom_name,start_record,end_record*) ··· 指定した原子種 **Atom_Name** の速度自己相関関数を求める
  - *atom_name* ··· 自己相関関数を計算する原子名 **Atom_Name**
  - *start_record* ··· サンプリングの開始レコード No.
  - *end_record* ··· サンプリングの終了レコード No.

  return ··· **List** of $(Time, Cvel, (Cvel_x, Cvel_y, Cvel_z))$

- **molVelocityAutoCorrelation**(*mol_name,start_record,end_record*) ··· 指定した分子名 **Mol_Name** の重心速度の自己相関関数を求める
  - *mol_name* ··· 自己相関関数を計算する分子名 **Mol_Name**
  - *start_record* ··· サンプリングの開始レコード No.
  - *end_record* ··· サンプリングの終了レコード No.

  return ··· **List** of $(Time, Cvel, (Cvel_x, Cvel_y, Cvel_z))$

- **forceAutoCorrelation**(*atom_name,start_record,end_record*) ··· 指定した原子種 **Atom_Name** に作用する力の自己相関関数を求める
  - *atom_name* ··· 自己相関関数を計算する原子名 **Atom_Name**
  - *start_record* ··· サンプリングの開始レコード No.
  - *end_record* ··· サンプリングの終了レコード No.

  return ··· **List** of $(Time, Cforce, (Cforce_x, Cforce_y, Cforce_z))$

- **stressAutoCorrelation**(*atom_name,start_record,end_record*) ··· 系のストレスの自己相関関数を求める
  - *start_record* ··· サンプリングの開始レコード No.
  - *end_record* ··· サンプリングの終了レコード No.

  return ··· **List** of $(Time, Cpress, (Cstress_x, Cstress_y, Cstress_z), (Cstress_x, Cstress_y, Cstress_z))$

7.4.5 CognacFileConvert.py

クラス

CognacFileConvert(*udffile,record*)

- *udffile* ··· オープンする UDF file name
- *record* ··· オープンする Record No.

メソッド

- **writePDB**(*file,scale*) ··· Protein Data Bank(PDB) format に変換しファイル出力
  - *file* ··· Output file name
  - *scale* ··· Scale factor
    実単位系に変換する場合など UDF の座標値に scale factor をかけた値を出力する (default=1.0)。例えば単位長さ 1σ=4.0Å で有るようなモデルで **COGNAC** の計算を行った場合、Scale factor=4.0 に設定すると Å 単位の長さに変換して出力される。
- **writeCAR**(*file,scale*) ··· Accelrys car file format に変換しファイル出力
  - *file* ··· Output file name
  - *scale* ··· Scale factor
    実単位系に変換する場合など UDF の座標値に scale factor をかけた値を出力する (default=1.0)
- **writeXYZ**(*file,scale*) ··· XYZ format に変換しファイル出力
  - *file* ··· Output file name
  - *scale* ··· Scale factor
    実単位系に変換する場合など UDF の座標値に scale factor をかけた値を出力する (default=1.0)
- **writeAVS**(*file*) ··· **Grid_Density** データを AVS で読み込まれる規則格子データ format に変換しファイル出力
  - *file* ··· Output file name
- **write**(*type,file,scale*) ··· 指定された format に変換しファイル出力
  - *type* ··· Output file format **'pdb'/'car'/'XYZ'/'AVS'**
  - *file* ··· Output file name
  - *scale* ··· Scale factor
    実単位系に変換する場合など UDF の座標値に scale factor をかけた値を出力する (default=1.0)。*type*=**'AVS'** の場合、このパラメータは使われない。

7.4.6 CognacUtility.dll/so

解析用スクリプトで利用する、境界条件による座標変換、自己相関関数等を高速に行うためのライブラリ。入出力は UDF に依存しない

関数群

- 境界条件の操作に関する関数群
  - **setCell**(*arg1*) ··· Unit cell のセット
    - *arg1* ··· Tuple of unit cell data, **(a,b,c,alpha,beta,gamma)**.
      角度は degree で与えること

return ··· **None**

– **getCell()** ··· セットされた Unit cell を返す
return ··· Tuple of unit cell data

– **getH()** ··· セットされた Unit cell より計算される Conversion matrix **H** を返す
$\mathbf{r} = \mathbf{H}\mathbf{r}'$, **r**:real coordinate, $\mathbf{r}'$:normalized coordinate
return ··· **H**

– **getHinv** ··· セットされた Unit cell より計算される Conversion matrix $\mathbf{H}^{-1}$ を返す
$\mathbf{r}' = \mathbf{H}^{-1}\mathbf{r}$
return ··· $\mathbf{H}^{-1}$

H および $\mathbf{H}^{-1}$ は対称性を考慮し **(xx,yy,zz,yz,xz,xy)** の形式で返る

– **getVolume()** ··· セットされた Unit cell より計算される Volume を返す
return ··· volume

– **setBoundary(***arg1***)** ··· Boundary conditions のセット

  * *arg1* ··· Tuple of boundary conditions, **(X,Y,Z)**.
    X,Y,Z は境界条件を区別する integer
    0(None)/1(Periodic)/2(Reflective1)/3(Reflective2)

return ··· **None**

– **getBoundary()** ··· セットされた Boundary conditions を返す
return ··· Tuple of boundary conditions, **(X,Y,Z)**.

– **distanceWithBoundary(***arg1,arg2***)** ··· 2 点間の、ミニマムイメージの差ベクトルを返す

  * *arg1/arg2* ··· Tuple of position **(x,y,z)** of each points

return ··· Tuple of differential vector **(dx,dy,dz)**

– **positionWithBoundary(***arg1***)** ··· 与えられた座標のユニットセル内 $0 \leq x, y, z < Cell_{Max}$ に存在するイメージの座標を返す

  * *arg1* ··· Tuple of position **(x,y,z)**

return ··· Tuple of position in unit cell **(x',y',z')**

– **getShift(***arg1***)** ··· 与えられた座標をユニットセル内の座標に変換する際のセルサイズのシフト数を返す

  * *arg1* ··· Tuple of position **(x,y,z)**

return ··· Tuple of shift **(shift X, shift Y, shift Z)**
ユニットセル内の座標 **r'** は $\mathbf{r}' = \mathbf{H}(\mathbf{H}^{-1}\mathbf{r} - \mathbf{shift})$ で求められる

- **positionWithShift(***arg1,arg2***)** ··· 座標とシフトベクトルを与えて、シフト後の座標を返す
  - ∗ *arg1* ··· Tuple of position **(x,y,z)**
  - ∗ *arg2* ··· Tuple of shift **(shift x,shift y,shift z)**

  return ··· Tuple of shifted position **(x',y',z')**

- データ列のセットに関する関数群

  後述のようなデータ列の解析を行うために、あらかじめデータをストアしておく必要がある。
  ライブラリの内部では、これらのデータは **map**⟨**string,vector**⟨**double**⟩ ⟩ あるいは
  map⟨**string,vector**⟨**Vector3d**⟩ ⟩ (**Vector3d** とは **double x,y,z** をメンバーに持つクラス) の形でストアされる。これら **map** の操作のための Python interface として以下のメソッドが提供されている。

  - **pushScalar(***arg1,arg2***)** ··· Scalar data をストアする
    内部的には Key で指定する **vector** に push_back していく
    - ∗ *arg1* ··· Key (string)
    - ∗ *arg2* ··· Scalar data

    return ··· **None**

  - **pushVector(***arg1,arg2***)** ··· 3D-Vector data をストアする
    内部的には Key で指定する **vector** に push_back していく
    - ∗ *arg1* ··· Key (string)
    - ∗ *arg2* ··· tuple of 3D-Vector data (x,y,z)

    return ··· **None**

  - **sizeScalar(***arg1***)** ··· Key で指定する **vector**⟨**double**⟩ の要素の数を返す
    - ∗ *arg1* ··· Key (string)

    return ··· 要素数

  - **sizeVector(***arg1***)** ··· Key で指定する **vector**⟨**Vector3d**⟩ の要素の数を返す
    - ∗ *arg1* ··· Key (string)

    return ··· 要素数

  - **clearScalarMap()** ··· scalar **map** をすべてクリアする
    return ··· **None**
  - **clearVectorMap()** ··· Vector3d **map** をすべてクリアする
    return ··· **None**

- データ列の解析に関する関数群

  - **pairDistribution(***arg1,arg2,arg3,arg4,arg5,arg6***)** ··· Pair distribution を計算する

* *arg1* ··· Vector **map** の Key List の tuple **([key1])** or **([key1,key2])**
  key が一つの場合 (A=B)A-A の分布、2 つの時 (A≠B)A-B の分布を計算する
* *arg2* ··· ヒストグラムの切片数,bin
* *arg3* ··· ヒストグラム切片のサイズ, width
* *arg4* ··· 分子内のみ $intra = 1$、分子間のみ $intra = -1$ あるいはすべての Pair$intra = 0$ を対象にする
* *arg5* ··· 扇状平均をとる場合の最小角度, minangle
* *arg6* ··· 扇状平均をとる場合の最大角度, maxangle

return ··· List of Probability, **[gr,gr_x,gr_y,gr_z]**
Probability は、key1 で指定したデータ列 1 データあたりの、切片 i の距離範囲 ($i * width < r < (i+1) * width$) に存在する key1 あるいは key2 の個数を返す。**gr_x,gr_y,gr_z** は各々x 軸、y 軸、z 軸を中心として最小角度、最大角度で指定される範囲に存在する Pair の分布を返す

– **msd(***arg1***)** ··· 平均自乗変位 MSD を計算する

$$MSD(|i-j|) =< (\mathbf{V}(i) - \mathbf{V}(j))^2 > \tag{7.1}$$

V(i):i-th data in Vector map

* *arg1* ··· Vector **map** の Key
  arg1 を与えない場合、ストアされているすべての Vector data から MSD を計算する

return ··· List of [**MSD_x,MSD_y,MSD_z**]

– **scalarCorrelation(***arg1,arg2***)** ··· Scalar data の相関関数 C_s を計算する

$$C_s(|i-j|) =< S(i) * S(j) > \tag{7.2}$$

S(i):i-th data in scalar map

* Usage1: 引数を与えない場合
  Scalar **map** にストアされているすべてのデータの Auto correlation を計算する
* Usage2: *arg1* のみ与えた場合
  arg1 を Key としてストアされている Scalar data の Auto correlation を計算する
* Usage3: *arg1/arg2* を与えた場合
  arg1/arg2 を Key としてストアされている Scalar data の Cross correlation を計算する

return ··· List of correlation

– **vectorCorrelation(***arg1,arg2***)** ··· Vector data の相関関数 C_v を計算する

$$C_v(|i-j|) =< \mathbf{V}(i) * \mathbf{V}(j) > \tag{7.3}$$

V(i):i-th Vector data in vector map

* Usage1: 引数を与えない場合
  Vector **map** にストアされているすべてのデータの Auto correlation を計算する
* Usage2: *arg1* のみ与えた場合
  arg1 を Key としてストアされている Vector data の Auto correlation を計算する

* Usage3: *arg1/arg2* を与えた場合
  arg1/arg2 を Key としてストアされている Vector data の Cross correlation を計算する

return ··· List of [**corr_x,corr_y,corr_z**]

使用例

CognacUtility の関数の簡単な使用例を示す。さらに具体的な解析における使用法は “CognacGeometryAnalysis.py/CognacTrajectoryAnalysis.py” を参照のこと。

1. 境界条件関連の関数群の使用例

```
%
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 *          # モジュールのインポート
>>> cell = (10,10,10,90,90,90)           # Unit cell のセット
>>> setCell(cell)
>>> boundary = (1,1,1)                   # 境界条件のセット
>>> setBoundary(boundary)                # すべて周期境界にセットする
>>> print getVolume()                    # ユニットセルの体積を計算
1000.0
>>> pos1 = (5,15,-5)
>>> pos2 = (7,13,-9)
>>> print positionWithBoundary(pos1)     # pos1 のユニットセル内のイメージ
(5.0, 5.0, 5.0)                          # 座標を計算
>>> print positionWithBoundary(pos2)
(7.0, 3.0, 1.0)
>>> print distanceWithBoundary(pos1,pos2)# pos1/pos2 の最短ベクトルを
(2.0, -2.0, -4.0)                        # 計算
>>> shift = getShift(pos1)               # pos1 のユニットセル内のイメージ
>>> print shift                          # 求めるためのシフト量を計算
(0.0, 1.0, -1.0)
>>> print positionWithShift(pos2,shift)  # shift を使って pos2 を移動
(7.0, 3.0, 1.0)
>>>
```

2. 相関関数等の使用例

```
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 *           # モジュールのインポート
>>> from math import *                    # pi を使うため math モジュールをイ
                                          # ンポート
```

```
>>> for i in range(0,10):                # "A"/"B"を key とするスカラーデー
    pushScalar("A",i)                    # タ列をセット
    pushScalar("B",i*i)                  # map["A"]=[0,1,2,3...]
                                         # map["B"]=[0,1,4,9...]
>>> corrAA = scalarCorrelation("A")      # map["A"] の自己相関関数を計算
>>> corrAB = scalarCorrelation("A","B")  # map["A"]/map["B"] 間の相互相関
                                         # 関数を計算
>>> corrAll = scalarCorrelation()        # すべてのスカラーマップの平均自
                                         # 己相関関数を計算
>>> for i in range(0,len(corrAA)):       # 結果を|i-j|=0,1,2... の順で表示
    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):               # 立方格子上に"pos"という key で
            for j in range(0,10):        # データをセット
                for k in range(0,10):
                        pushVector("pos",(i,j,k))


>>> setCell( (10,10,10,90,90,90) )       # Unit cell と境界条件のセット
>>> setBoundary( (1,1,1) )
>>>
>>> bin = 50
>>> width = 0.1
>>> pr = pairDistribution( tuple(["pos"]), bin, width)  # 分布関数の計算
>>>
>>> for i in range (0,bin):              # 球面の表面積で割り動径分布を
                                         # 計算
            r = (i+0.5)*width
            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
...
（略）

>>>
>>> clearVectorMap()                    # Vector map をクリアする
>>> for i in range(0,10):               #
    pushVector( "pos1", (i,i*2,i*3) )   # map["pos1"],map["pos2"] に
    pushVector( "pos2", (i,i/2,i/3) )   # 適当な値をセット

>>> gpos1 = msd("pos1")                 # pos1 の MSD を計算
>>> gpos2 = msd("pos2")                 # pos2 の MSD を計算
>>> gave = msd()                        # pos1/pos2 平均の MSD を計算
>>> for i in range(0,len(g1ave)):       # 結果の出力
    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　Action 機能による解析ツールの利用

COGNAC には上記の解析用スクリプトを簡便に利用するために、**GOURMET** の持つ **Action** 機能を用いた解析コマンドが用意されている。主なコマンドは File name が表示される UDF ツリーのルート部分を右クリックすることにより現れる、図 7.1 に示すようなコマンドリストから選択することにより起動することが出来る。

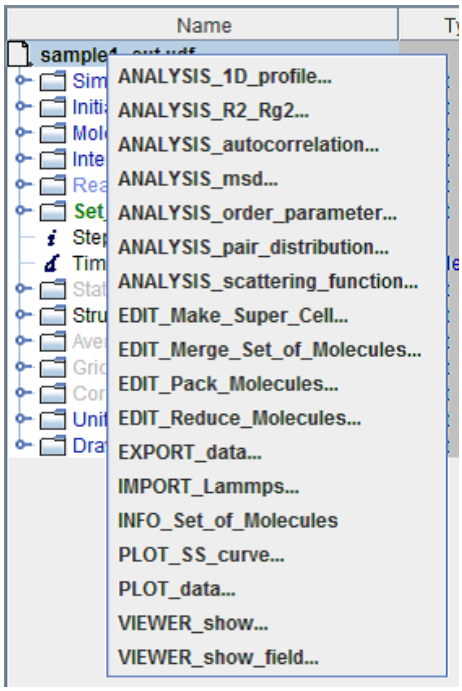


図 7.1: **Action** により表示されたコマンドリストの例

Action コマンドの内容、与えるパラメータは Python メソッドおよび引数に準じている。以下に **COGNAC** udf において利用できる **Action** command をあげる。

- **ANALYSIS_1D_profile...**

  1D profile の計算とプロット。ルートアイコンより起動できる。図 7.2 に示す引数の内容は以下の通り。

  - **atom_or_molecule** ··· **atom/molecule** の選択
  - **name_list** ··· Density profile を計算する **{Atom|Mol}_Name** のリスト。（例）**['A','B']**
  - **direction** ··· 軸の指定。**z/y/x** より選択する。

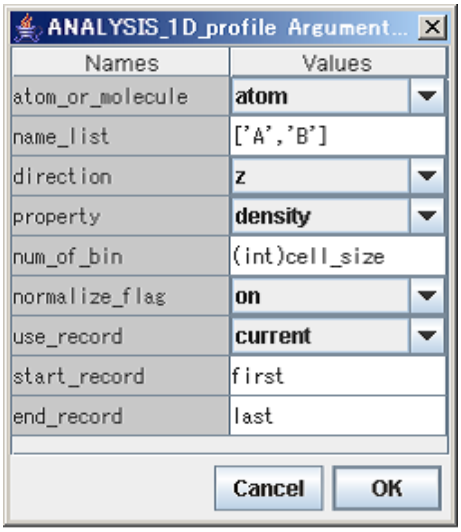


図 7.2: **ANALYSIS_1D_profile...** の引数

- **property** ··· 解析するプロパティー。**density/velocity** より選択する。
- **num_of_bin** ··· 分割数、整数を入力する。デフォルトでは **(int)cell_size** が指定される。
- **normalize_flag** ··· 規格化するかどうかのフラグ。**on/off** より選択する。
- **use_record** ··· 解析対象のレコードの選択。**current/range** より選択。**current** の場合現在選択されている Record を対象とする。**range** の場合、Record の範囲を指定して、その間の平均を計算する。
- **start_record** ··· **use_record** において **range** を選択した場合、選択する先頭の Record No. を指定
- **end_record** ··· **use_record** において **range** を選択した場合、選択する最終の Record No. を指定

- **ANALYSIS_R2_Rg2...**

  指定した分子の、末端間距離自乗 R^2 および慣性半径自乗 Rg^2 の計算。ルートアイコンより起動できる。図 7.3 に示す引数の内容は以下の通り。

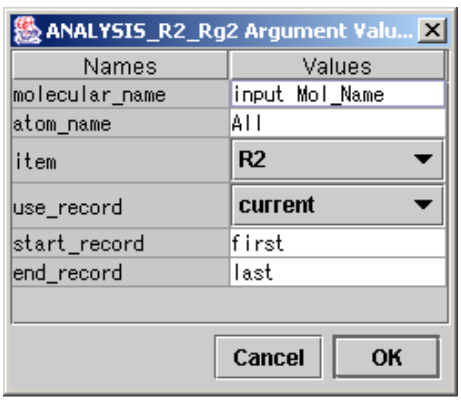


図 7.3: **ANALYSIS_R2_Rg2...** の引数

- **molecular_name** ··· **Mol_Name** の指定。デフォルトでは **Set_of_Molecules.molecule[0]** の **Mol_Name** がセットされる

- **atom_name** ··· 指定した **Atom_Name** のみの R^2、Rg^2 を計算する。
  ブロックコポリマーの部分鎖の形状の計算等に有効。
  'All' を指定した場合、分子全体で計算。
- **item** ··· 計算する項目。**R2/Rg2/both** より選択する
- **use_record** ··· 解析対象のレコードの選択。**current/range** より選択。**current** の場合現在選択されている Record を対象とする。**range** の場合、Record の範囲を指定して、Record 毎の変化を計算する。
- **start_record** ··· **use_record** において **range** を選択した場合、選択する先頭の Record No. を指定
- **end_record** ··· **use_record** において **range** を選択した場合、選択する最終の Record No. を指定

use_record において **current** を指定した場合は、**Mol_Name** で指定した分子すべての R^2、Rg^2 のリストが Python log window に出力される。
use_record において **range** を指定した場合は、**Mol_Name** で指定した分子すべての R^2、Rg^2 の各レコードにおける平均値が Python log window に出力される。

- **ANALYSIS_autocorrelation...**

自己相関関数の計算とプロット。ルートアイコンより起動できる。図 7.4 に示す引数の内容は以下の通り。

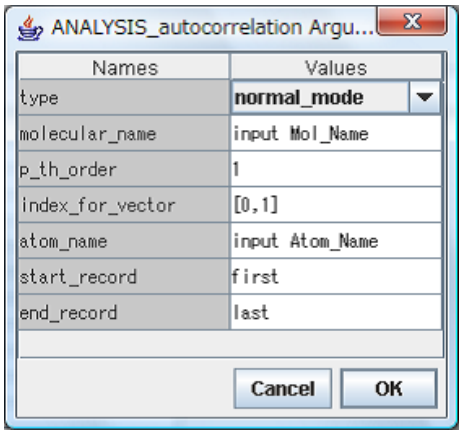


図 7.4: **ANALYSIS_autocorrelation...** の引数

- **type** ··· 自己相関関数を計算する物性。**normal_mode/vector/atom_velocity/mol_velocity /force/stress** より選択する。
- **molecular_name** ··· **Mol_Name** の指定。デフォルトでは **Set_of_Molecules.molecule[0]** の **Mol_Name** がセットされる
- **p_th_order** ··· Normal mode のオーダー。**type** で **normal_mode** を選択した時のみ有効。
- **index_for_vector** ··· 自己相関関数を計算するベクトルを定義する Atom Index のペア **type** で **vector** を選択した時のみ有効。
- **atom_name** ··· 速度自己相関関数を計算する Atom Name。**type** で **atom_velocity** あるいは **force** を選択した時のみ有効。
- **start_record** ··· サンプリングの開始レコード No.。デフォルト (**first**) は **Record 0**。

– **end_record** ··· サンプリングの終了レコード No.。デフォルト (**last**) は最終レコード。
start_record,**end_record** ともにデフォルト値の場合は **Time** を持つレコードのみを抽出し、**Record** ＝ **“Initial”** などのレコードは読み飛ばされる。

- **ANALYSIS_msd...**

平均自乗変位 (MSD) の計算とプロット。ルートアイコンより起動できる。図 7.5 に示す引数の内容は以下の通り。

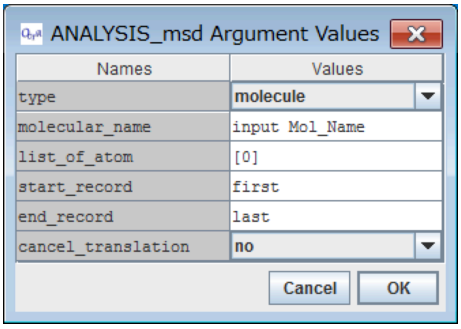


図 7.5: **ANALYSIS_msd...** の引数

– **type** ··· MSD を計算する対象の指定。**molecule/atom** より指定。
– **molecular_name** ··· **Mol_Name** の指定。デフォルトでは **Set_of_Molecules.molecule[0]** の **Mol_Name** がセットされる
– **list_of_atom** ··· MSD を計算する atom の list。**type** において **atom** を選択した時のみ有効。
– **start_record** ··· サンプリングの開始レコード No.。デフォルト (**first**) は **Record 0**。
– **end_record** ··· サンプリングの終了レコード No.。デフォルト (**last**) は最終レコード。
start_record,**end_record** ともにデフォルト値の場合は **Time** を持つレコードのみを抽出し、**Record** ＝ **“Initial”** などのレコードは読み飛ばされる。
– **cancel_translation** ··· 系全体の重心の並進の寄与を補正するかどうかのフラグ。**no/yes** から選択する。

- **ANALYSIS_order_parameter...**

配向秩序パラメータの計算。ルートアイコンより起動できる。図 7.6 に示す引数の内容は以下の通り。

– **molecule_or_bond** ··· 分子の配向か bond の配向かの選択。**molecule** を選択した場合、以下の **molecular_name**, **order_vector** および **vector_definition** で対象とする分子名、Order parameter を計算する分子内のベクトルを指定する。**bond** を指定した場合これらの入力値は用いられず、**Set_of_Molecules** に含まれる全ての **Potential_Name** の bond の Order parameter が計算される
– **molecular_name** ··· **Mol_Name** の指定。デフォルトでは **Set_of_Molecules.molecule[0]** の **Mol_Name** がセットされる
– **order_vector** ··· Order parameter を計算するベクトルの指定。**Inertia axis 0(minimum moment)/ Inertia axis 1/ Inertia axis 2(maximum moment)/ Vector between atoms** より選択する。
– **vector_definition** ··· Order parameter を計算するベクトルを指定する Atom Index のペア。**order_vector** において、**Vector between atoms** を指定した場合のみ有効

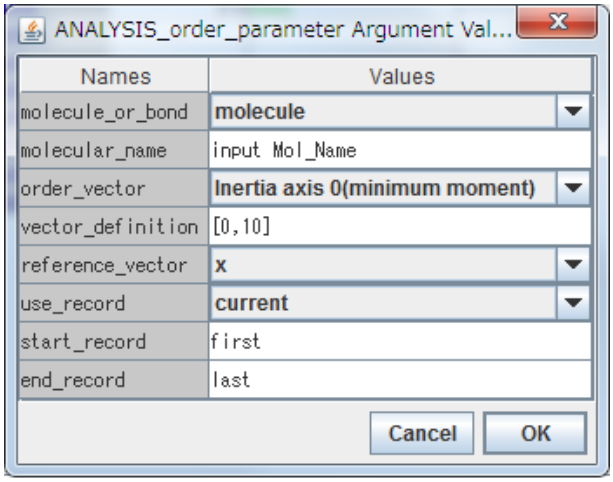


図 7.6: **ANALYSIS_order_parameter...** の引数

- **reference_vector** ··· Order parameter を計算するための参照ベクトルの指定。**x/y/z/average** より選択する。**x**、**y** あるいは **z** が選択された場合は、各々の座標軸が参照ベクトルに指定される。**average** が指定された場合は、**order_vector** で指定されるベクトルの総分子の平均、あるいは bond ベクトルの平均値が参照ベクトルに指定される。
- **use_record** ··· 解析対象のレコードの選択。**current/range** より選択。**current** の場合現在選択されている Record を対象とする。**range** の場合、Record の範囲を指定して、Record 毎の変化を計算する。
- **start_record** ··· **use_record** において **range** を選択した場合、選択する先頭の Record No. を指定
- **end_record** ··· **use_record** において **range** を選択した場合、選択する最終の Record No. を指定

molecule_or_bond にて **molecule** を選択した際、**use_record** において **current** を指定した場合、**Mol_Name** で指定した分子すべての Order parameter のリストが Python log window に出力される。**use_record** において **range** を指定した場合は、**Mol_Name** で指定した分子すべての Order parameter の各レコードにおける平均値が Python log window に出力されるとともにプロットされる。

一方 **molecule_or_bond** にて **bond** を選択した際、**use_record** において **current** を指定した場合、すべての bond の Order parameter が **Set_of_Molecules** に含まれる **Potential_Name** 毎にリストとして Python log window に出力される。
use_record において **range** を指定した場合は、同一の **Potential_Name** を持つ bond の Order paramter の各レコードにおける平均値が Python log window に出力されるとともにプロットされる。

- **ANALYSIS_pair_distribution...**

Pair distribution function の計算およびプロット。ルートアイコンより起動できる。図 7.7 に示す引数の内容は以下の通り。

- **atom_or_molecule** ··· Atom あるいは Molecule の選択。Molecule を選択した場合は分子の重心の Pair distribution function を計算する
- **name_list** ··· Pair distribution function を計算する **Atom_Name** あるいは **Mol_Name** のリスト。（例）**['A','B']**

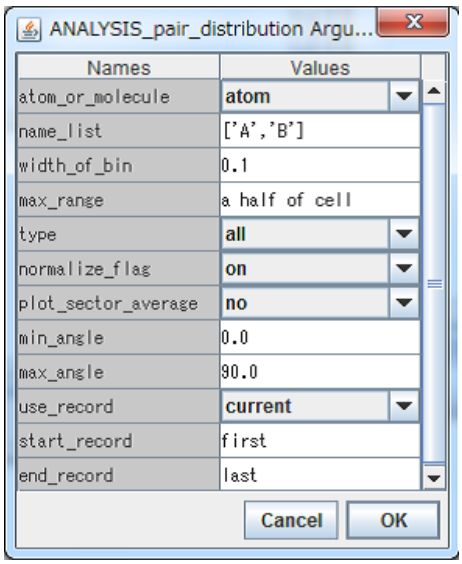


図 7.7: **ANALYSIS_pair_distribution...** の引数

- **width_of_bin** ··· 分割幅。
- **max_range** ··· 最大距離。デフォルトは最小ユニットセルの 1/2。
- **type** ··· 動径分布を計算する対象の指定**'all'/'inter'/'intra'** より選択する。
  'all の場合、**name_list** で指定するすべてのペアの動径分布を計算する。
  'inter' の場合、分子間のペアのみ、
  'intra' の場合、分子内のペアのみの動径分布を計算する
- **normalize_flag** ··· 規格化するかどうかのフラグ。**on/off** より選択する。
- **plot_sector_average** ··· x,y,z 軸を中心とした扇状平均を出力するかどうかのフラグ。**on/off** より選択する。
- **min_average** ··· 扇状平均をとる最小角度
- **max_average** ··· 扇状平均をとる最大角度
- **use_record** ··· 解析対象のレコードの選択。**current/range** より選択。**current** の場合現在選択されている Record を対象とする。**range** の場合、Record の範囲を指定して、その間の平均を計算する。
- **start_record** ··· **use_record** において **range** を選択した場合、選択する先頭の Record No. を指定
- **end_record** ··· **use_record** において **range** を選択した場合、選択する最終の Record No. を指定

- **ANALYSIS_scattering_function...**

  グリッドポイントの濃度分布より散乱関数を計算、プロットする。ルートアイコンより起動できる。図 7.8 に示す引数の内容は以下の通り。

  - **UseGridDensity** ··· **COGNAC** の出力に含まれる **Grid_Denisty** を用いるか、新たに濃度分布を計算して散乱関数を計算するかの選択。**'yes'/'no'** より選択する。
    Grid_Density は **COGNAC** 起動時に **Simulation_Conditions.Density_Output** で指定した場合のみ出力される。

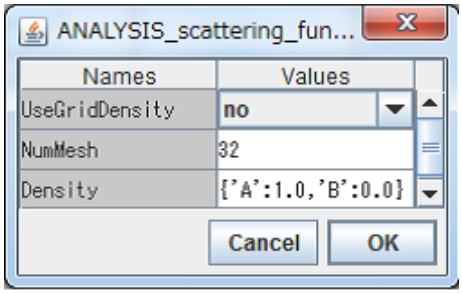


図 7.8: **ANALYSIS_scattering_function...** の引数

- **NumMesh** ··· **UseGridDensity = 'no'** の場合、分割するメッシュ数の指定。整数 n を指定した場合、全ての軸は同数で分割される。要素数 3 のリスト *[nx,ny,nz]* で指定した場合、各軸指定された数で分割される。
  UseGridDensity = 'yes' の場合はこの指定は無視される。
- **Density** ··· **Atom_Name** の密度の計算に対する寄与を与える。
  （例）**{'A':1.0,'B':0.0}** の場合、**Atom_Name = 'A'** の分布に基づいて、濃度分布が与えられる。
- **use_record** ··· 解析対象のレコードの選択。**current/range** より選択。**current** の場合現在選択されている Record を対象とする。**range** の場合、Record の範囲を指定して、その間の平均を計算する。
- **start_record** ··· **use_record** において **range** を選択した場合、選択する先頭の Record No. を指定
- **end_record** ··· **use_record** において **range** を選択した場合、選択する最終の Record No. を指定

- **EXPORT_data...**

  ファイルフォーマットの変換。現在表示しているレコードのデータを出力する。ルートアイコンより起動できる。図 7.9 に示す引数の内容は以下の通り。

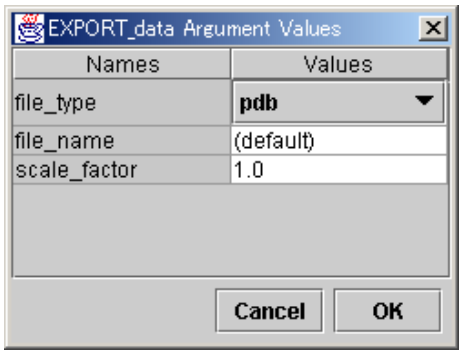


図 7.9: **EXPORT_data...** の引数

- **file_type** ··· 変換するフォーマット。**udf/pdb/car/XYZ/LAMMPS/AVS** より選択する。以下に各フォーマットの内容を概説する。
  - **udf** ··· **udf** を選択した場合は、フォーマットの変更は無しに、1 レコードのみを抽出した udf ファイルを新たに作成する。
  - **pdb** ··· **Structure** データを Proten Data Bank 形式で出力する。CONECT データもあわせて作成される

  - **car** ··· **Structure** データを Accelrys 社 car ファイル形式で出力する。
  - **XYZ** ··· **Structure** データを XYZ 形式で出力する。
  - **LAMMPS** ··· シミュレーションモデルの情報を並列化分子動力学プログラム LAMMPS Data 形式で出力する。LAMMPS およびそのデータ形式に関しては Web ページ (http://lammps.sandia.gov/) を参照 のこと
  - **AVS** ··· **Grid_Density** データを AVS 構造格子形式で出力する。

- **file_name** ··· 出力するファイル名。デフォルトでは UDF file 名のプリフィックスにレコード No. を追加し、拡張子を変更した名前で出力する。
- **scale_factor** ··· 出力座標の Scale factor。**file_type=udf** あるいは **AVS** の場合、このパラメータは使われない。

- **IMPORT_LAMMPS...**

  並列化分子動力学プログラム LAMMPS のトラジェクトリ出力 (dump file) を時系列に従い新たな **Record** を作成し、**Strucutre** データに読み込む。LAMMPS およびそのデータ形式に関しては Web ページ (http://lammps.sandia.gov/) を参照 のこと

  ルートアイコンより起動できる。図 7.10 に示す引数の内容は以下の通り。

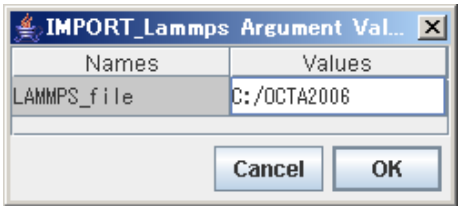


図 7.10: **IMPORT_LAMMPS...** の引数

  - **LAMMPS_file** ··· LAMMPS dump ファイル名を指定す

- **INFO_Set_of_Molecules**

  Mol_Name、分子鎖長等の **Set_of_Molecules** の情報を出力する。ルートアイコンより起動できる。

- **EDIT_Make_Super_Cell...**

  ユニットセルをコピーして、大きなサイズのセル (Super_Cell) を作成する。ルートアイコンより起動できる。

  図 7.11 に示す引数の内容は以下の通り。

  - **nx, (ny, nz)** ··· 各辺の拡大する倍数

- **EDIT_Merge_Set_of_Molecules...**

  指定した UDF file の **Set_of_Molecules** と **Structure** データを、現在の UDF にマージする。

  ルートアイコンより起動できる。図 7.12 に示す引数の内容は以下の通り。

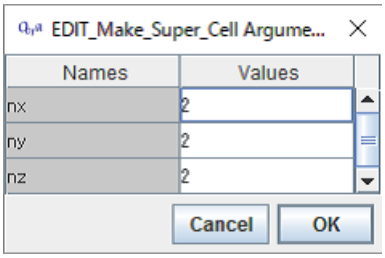


図 7.11: **EDIT_Male_Super_Cell. . .** の引数

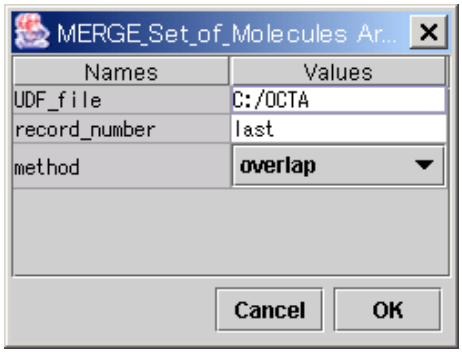


図 7.12: **EDIT_Merge_Set_of_Molecules. . .** の引数

- **UDF_file** · · · 現在の UDF データにマージする UDF ファイル名
  入力欄を右クリックすると、ファイルブラウザが現れる
- **record_number** · · · マージする UDF ファイルの Record number1
  Record number(int) あるいは予約語 **last**（最終レコードをマージする）を入力する
- **method** · · · マージする方法
  overlap/x_layer/y_layer/z_layer より選択。
  overlap を選択すると、現在の UDF のユニットセルに全ての Atom がパックされる。
  x(,y,z)_layer を選択すると、指定した方向に層構造を生成する。指定した方向のユニットセルサイズは、マージする 2 つのユニットセルサイズの和になる。

なお、このデータ編集は **GOURMET** 上の UDF データに行われ、**Save** するまでは UDF file には反映されない。

(注)x(,y,z)_layer オプションで層構造を作成する際、個々の層の積層する方向の atom の座標は、セル内に収まっていないと分離した層構造にならない。周期境界やスタガード反射境界条件の際に初期座標をランダムに発生させると、ユニットセルの外側にも座標を与えることがあるので、境界条件は **None** にして作成する必要がある。

- **EDIT_Pack_Molecules**

  個々の分子の重心座標をユニットセル内に平行移動を行う。Polymerization の機能により、Periodic bond が形成されている場合は Periodic bond を利用しないように個々の Atom の座標がシフトされる。

  ルートアイコンより起動できる。図 7.13 に示す引数の内容は以下の通り。

  - **continue** · · · コマンドを実行するかどうかの確認。**Yes** を指定すると座標変換が実行される

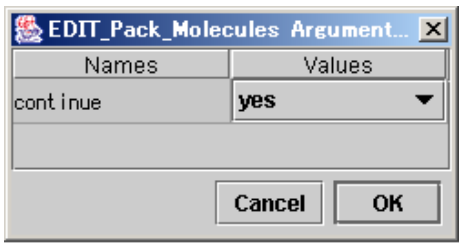


図 7.13: **EDIT_Pack_Molecules...** の引数

なお、このデータ編集は **GOURMET** 上の UDF データに行われ、**Save** するまでは UDF file には反映されない。

- **EDIT_Reduce_Molecules...**

  Set_of_Molecules と現在表示している **Structure** データを指定した密度になるように、分子数を減少させる。DPD の結果からビーズ－スプリングモデルへのズーミング等に有効な機能である。

  ルートアイコンより起動できる。図 7.14 に示す引数の内容は以下の通り。

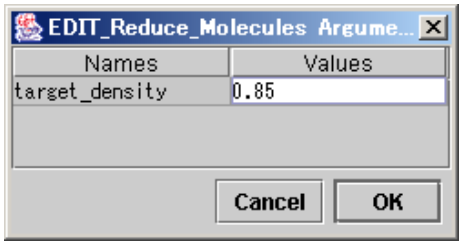


図 7.14: **EDIT_Reduce_Molecules...** の引数

  – **target_density** ··· ターゲットとする密度。現在の仕様では Atom の数密度でスケールする。

  なお、このデータ編集は **GOURMET** 上の UDF データに行われ、**Save** するまでは UDF file には反映されない。

- **PLOT_SS_curve...**

  Simple_Elongation5.2.2 の機能を用いて、ユニットセルの変形を行ったシミュレーション結果より、応力－歪 (S-S) カーブをプロットする。
  ルートアイコンより起動する。図 7.15 に示す引数の内容は以下の通り。

  – **mode** ··· シミュレーション時の実施した変形モードの指定
  伸張 (**forward**) あるいは圧縮 (**reverse**)。**forward** の場合初期レコード、**reverse** の場合最終レコードの c 軸のセル長が歪を計算する基準長さになる。

  – **P_0** ··· 静水圧 **P_0** の定義。**zero** の場合、**P_0**=0 として **Stress_zz** の絶対値をそのまま出力。**(xx+yy)/2** の場合、応力は **Stress_zz - 1/2*(Stress_xx + Stress_yy)** で定義された値が出力される。

  – **Output_Fraction** ··· **yes** の場合、ストレスの Bond および Non bond ポテンシャル由来の成分も表示する

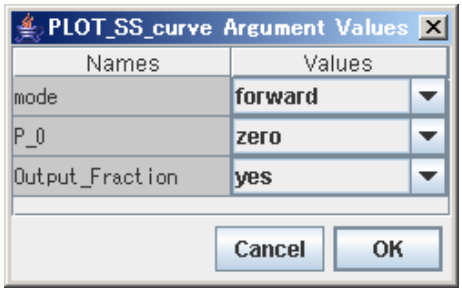


図 7.15: **PLOT_SS_curve...** の引数

- **PLOT_data...**

  指定した UDF location のデータをレコード順にプロットする。ルートアイコンより起動できる。図 7.16 に示す引数の内容は以下の通り。

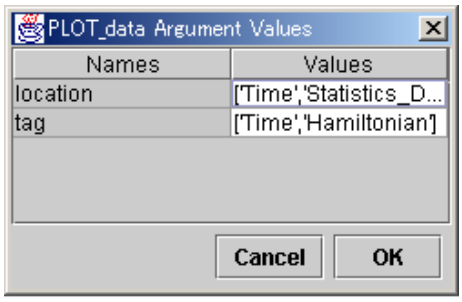


図 7.16: **PLOT_data...** の引数

  - **location** ··· プロットする UDF Location 名のペア。
    （例）**['Time','Statistics_Data.Energy.Batch_Average.Hamiltonian']** 一番目の Location を x 軸に、二番目を y 軸に取りプロットする。
  - **tag** ··· 各 Location につけるタグ。

- **VIEWER_show...**

  分子構造の **Viewer** window への表示。ルートアイコンより起動できる。図 7.17 に示す引数の内容は以下の通り。

  - **type** ··· 描画タイプ **line/ball-stick/rod/volume** より選択する。
  - **bc** ··· 描画時の境界条件の設定。**mol**：分子の重心がユニットセル内に位置するように境界条件を作用させて表示する
    atom：各原子がすべてユニットセル内に位置するように境界条件を作用させて表示する
    off：境界条件を考慮せず、Atom の座標値で表示する
  - **color** ··· 描画時の色指定 **atom**：Atom type で色分けする
    bond：Bond type で色分けする
    mol：分子毎に色分けする。ただし OCTA がデフォルトで持つ色の設定には限りがあるので、多数の分子を表示するときは、同じ色が繰り返し使われる
    molname：Molecular name で色分けする

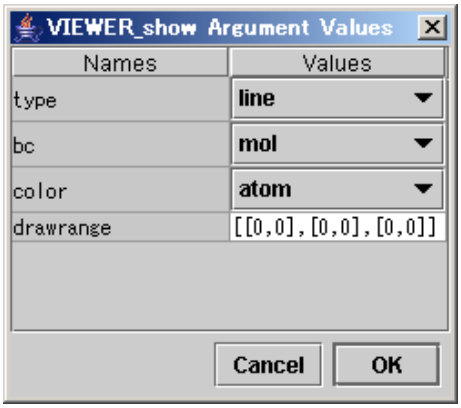


図 7.17: **VIEWER_show...** の引数

– **drawrange** ··· 描画時のユニットセル表示範囲
基本のユニットセルに含まれる分子に加えて、周囲に存在するイメージも表示することが出来る。表示する範囲はリスト形式で与え、以下の順で範囲を指定する。
[[*amin,amax*],[*bmin,bmax*],[*cmin,cmax*]]
amin,amax,bmin... には各々a,b,c 軸方向に表示するセルの範囲を整数で指定する。例えば [[-1,1],[-1,1],[-1,1]] と指定した場合、a,b,c 軸すべて、基本セル内の分子と、-1 および 1 シフトしたイメージ分子が表示される。

- **Viewer_show_field...**

密度分布の **Viewer** への表示。ルートアイコンより起動できる。図 7.18 に示す引数の内容は以下の通り。

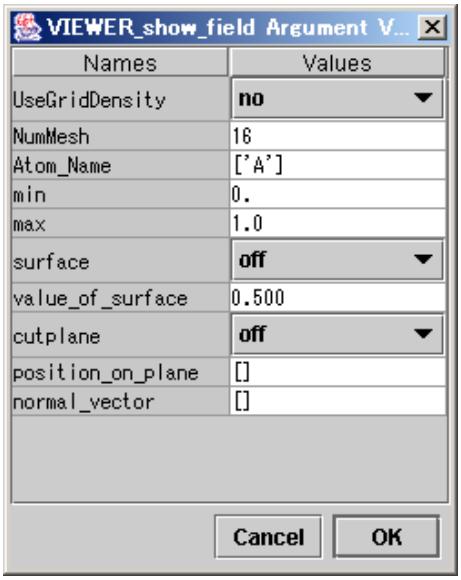


図 7.18: **VIEWER_show_field...** の引数

– **UseGridDensity** ··· UDF 中の Grid density のデータを用いるかどうかのフラグ。**yes/no** より選択する。**'yes'** を指定した場合、UDF に Grid density のデータが含まれている必要がある。**'no'** の場合は、Structure のデータより密度分布を計算する。

- **NumMesh** ··· 密度分布を計算する一辺あたりの格子点の数。**UseGridDensity** が **yes** の場合は、Structure のデータより Mesh データを読み込むのでこの値は無視される。
- **Atom_Name** ··· 密度を表示する Atom name の指定。
  Atom name のリストとして与える。
- **min** ··· カラーコンター表示の場合のスペクトルの最小値
- **max** ··· カラーコンター表示の場合のスペクトルの最大値
- **surface** ··· カラーコンター表示と等値面表示の選択フラグ。**yes/no** より選択する。
- **value_of_surface** ··· 等値面表示の場合の面の値
- **cutplane** ··· カラーコンター表示において、任意の面を指定する場合のフラグ。**yes/no** より選択する。**no** の場合、ユニットセルのすべての面が表示される。
- **position_of_plane** ··· カラーコンター表示において、任意の面を指定した場合の面の座標の指定。面を含む任意の座標を **[x,y,z]** の形式で指定する。
- **normal_vector** ··· カラーコンター表示において、任意の面を指定した場合の法線ベクトルの指定。**[x,y,z]** の形式で指定する。

- **show...**

  オブジェクト（分子、原子、結合）の **Viewer** window への表示。**Set_of_Molecules**、**molecule**、**atom** および **bond** において起動できる。**molecule**,**atom** および **bond** を選択する場合は、各オブジェクトの Index を指定して、**Action** を起動する。
  引数は **VIEWER_show...** のものとほぼ同様。

- **show_same_name...**

  同一の **Mol_Name** を持つ分子の **Viewer** window への表示。任意の molecule の Index が指定された場合、その molecule と同一の **Mol_Name** を持つ molecule がすべて表示される。
  引数は **VIEWER_show...** のものとほぼ同様。

- **show_same_type...**

  同一の **Atom_Type** を持つ原子団の **Viewer** window への表示。任意の atom の Index が指定された場合、その atom と同一の **Atom_Type** を持つ atom がすべて表示される。
  引数は **VIEWER_show...** のものとほぼ同様。

- **plot**

  Statistical_Data の経時変化のプロット。**Statistical_Data** 以下の各オブジェクトにおいて起動できる。**Temperature** のように単純な構造のものは、そのオブジェクトを選択することにより起動できる。一方、**Energy** のように複雑な構造を持つオブジェクトの場合、その下層の **Instantaneous**、**Batch_Average** あるいは **Total_Average** を選択することにより、個別にプロットを行うことができる。

- **{mol,atom,bond}_info**

  molecule、**atom** あるいは **bond** の持つ、名称やタイプなどの情報を Python log window に出力する。各オブジェクトの Index を選択することにより起動する。

以下の 3 つの **Action** コマンドは、**Viewer** window における atom の複数ピッキングにより、起動される。**GOURMET Viewer** window 内の **Picking** メニュー、あるいは Ctrl+M により複数ピッキングモードの on/off を行うことができる。詳細は「GOURMET 操作マニュアル」参照。

図 7.19 に 2 原子を選択した際の **Viewer** window の表示例を示す。この場合、**Distance...** コマンドがメニュー内に表示される。

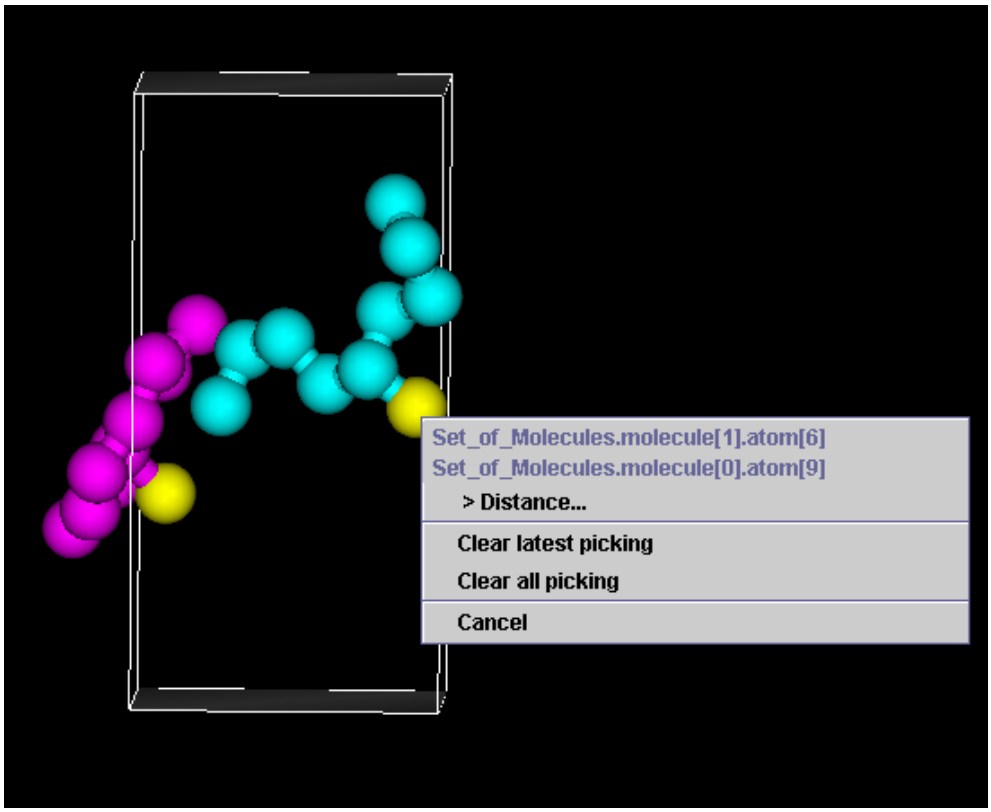


図 7.19: 複数ピッキングの例

3 あるいは 4 原子が選択された場合、各々**Angle** および **Torsion** がメニューに現れる。

- **Distance**

  選択された 2 原子間の距離を計算する。図 7.20 に示される引数の内容を以下に示す。

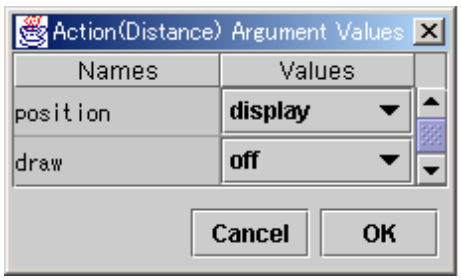


図 7.20: **Distance...** の引数

  – **position** ··· 境界条件の設定。**display/minimum** より選択
  display が選択された場合、**Viewer** window に表示されている 2 原子の座標間の距離が計算され

る。**minimum** が選択された場合、2 原子間の minimum image の距離が計算される。

– **draw** ··· **Viewer** window に距離を表示するかどうかのフラグ
図 7.21 に示すように、**Viewer** window 内に距離が表示される。
いずれの場合も計算結果は Python log window に表示される。

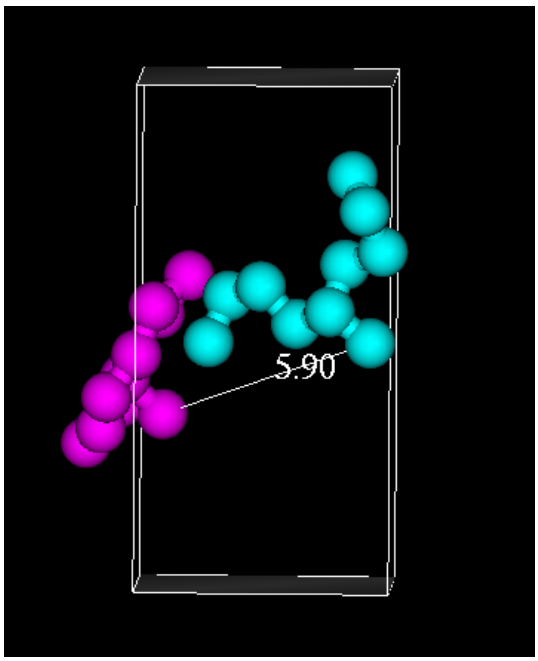


図 7.21: 距離表示の例

- **Angle**

  選択された 3 原子のなす角度を計算する。計算結果は Python log window に表示される。

- **Torsion**

  選択された 4 原子のなす二面角を計算する。計算結果は Python log window に表示される。

付 録A　コンパイル方法

ここでは付録 B で解説するようにユーザーポテンシャルを新たに定義した場合など、**COGNAC** のリコンパイルが必要になった際の方法を解説する。

COGNAC のコンパイル／リンクには **GOURMET** インターフェースライブラリ libplatform が必要になる。**OCTA8.3** において、提供される libplatform の内容は「プラットフォームインターフェースライブラリ libplatform リファレンスマニュアル」参照。

1. MinGW/Cygwin/Linux/Unix 環境におけるコンパイル

   “src” ディレクトリ内の “Makefile” を用いる。OS およびマシンタイプの判別は環境変数 **$PF_ENGINEARCH** に基づいて、“Makefile” の中で行われる。
   デフォルトでは OpenMP 並列版を make するが、非並列版を make する場合は Makefile 5 行目の ”OMP_FLAG =-fopenmp” をコメントアウトすればよい。

   手順

   (a) **makedepend** の実行（オプション）
   コンパイルする環境において **makedepend** コマンドが利用できる場合は、下記のように実行しておくとよい。

   ```
   % make depend
   ```

   （注意） “Makefile” にはオブジェクトファイルの dependency として”.cpp/.h” ファイルがセットされているが、その他の dependency をすべてセットするために **makedepend** を実行するとよい。これはプログラム開発を行い、ソースコードをしばしば編集する場合のみ必要で、既存のコードを単にコンパイル／リンクするためには必要ない。

   (b) コンパイル／リンク
   make のデフォルトターゲットにより自動的にコンパイル／リンクが行われる

   ```
   % make
   ```

   (c) インストール
   コンパイル／リンクして生成した実行ファイルを、実行ディレクトリにコピーする

   ```
   % make install
   ```

実行ファイル”cognac92{.exe}” は “$(PF_ENGINE)/bin/$(PF_ENGINEARCH)“ディレクトリにコピーされる。ここで**$PF_ENGINE**,**$PF_ENGINEARCH** は各々、OCTA に含まれるシミュレーションプログラム群をインストールしたディレクトリ、およびシミュレーションプログラムを実行する環境 (linux,cygwin etc.) を指定する環境変数で、あらかじめ設定されている必要がある。（詳細は「OCTA インストールマニュアル」参照）

（注意） いくつかの Unix 環境でコンパイルの確認は行われているが、make するためには **gmake** コマンドが必要になるので注意。各環境におけるコンパイルオプション等に関しては”Makefile” 参照。

2. Microsoft Visual C++ 2017 によるコンパイル

“src/objects/win_vc2017” ディレクトリにある “COGNAC.vcxproj” を開いて、コンパイル/リンクを行う。アクティブな構成によって”Win32”あるいは”x64”以下の “Release” あるいは “Debug” ディレクトリに実行ファイルができる。

（注意）本バージョンの COGNAC は Microsoft Visual C++ 2017 でのコンパイル、動作を確認しており、Visual C++ 旧バージョンのプロジェクトファイルは同梱されていない。Visual C++によるコンパイルが必要な場合はバージョン 2017 を利用のこと。

合わせて、GOURMET インターフェースライブラリ “libplatform.lib”も Visual C++ 2017 でコンパイルしたものを用意する必要があるので注意。詳細は「プラットフォームインターフェースライブラリ libplatform リファレンスマニュアル」参照

付 録B　システム拡張方法

本章では、**COGNAC** のシステム拡張方法として、相互作用ポテシンャル関数のユーザー定義の方法と使用法に関して解説する。

B.1　拡張できるポテシンャル

COGNAC においては以下の相互作用ポテシンャルにユーザー定義の関数形を追加することができる。

- Bond potential
- Angle potential
- Torsion potential
- Pair interaction
- External interaction

以下より、Bond potential を例にとり、新規なポテシンャル関数の作成と利用法を解説する。

B.2　用意すべきファイル

ユーザー定義の Bond potential を作成するためには以下のファイルが必要である。

```
userbond[1-3].h
userbond[1-3].cpp
```

現在の仕様としてユーザーポテンシャル関数は 3 種類まで作成、同時に使用することができ、各々 “userbond{1-3}.h/.cpp” というファイル名で用意しておく。ヘッダーファイルに関してはあらかじめ “include” ディレクトリに用意されているのでそれを修正すればよい。

ここでの 3 種類という制限はあくまで、関数形の種類で、同じ関数形でパラメータのみの異なるものに関しては、数は無制限に利用することができる。

Angle potential 等を作成する場合はファイル名の “bond” の部分を、“angle”、“torsion”、“pairinteraction”、“externalfield” としたファイルを作成する。

B.3　新たな Bond potential の追加法

B.3.1　userbond1.h/.cpp の例

以下に配布の “include” および “src” ディレクトリに含まれる “userbond1.h/.cpp” の例を示す。例に示す”userbond1” は harmonic potential と同様の $U_{bond} = 1/2k(r - r_0)^2$ で表される bond potential を定義する。

userbond1.h の例

```
///// 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_
```

userbond1.cpp の例

```
///// 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 メンバー変数の定義

“userbond1.h” の private メンバーにポテンシャル関数に用いる変数を書く。例では harmonic spring constant である **double kconst;** を定義している。

B.3.3 コンストラクタの引数および内容

“userbond1.h” あるいは”userbond1.cpp” においてコンストラクタの実装を書く。このコンストラクタにおいてポテンシャル関数の計算に必要な変数をすべて引数よりセットする。コンストラクタの引数は固定で変更の必要はなく、**params** というマップに実際に private 変数に設定したい変数が入って渡される。

この **params** というオブジェクトは STL のマップコンテナという型をもっており、通常の配列データのように **params**[**key**] という形でデータをとるが、ここで用いているマップは key の部分に string data を取って **params** のデータを選択する。例にあるように、**params**[**”K”**] と指定すると**”K”**という string データを key としてデータを取り出す。

また取り出した data も string 形のデータとしているので、double 型に変更して **kconst** というメンバーに代入したければ例に記すように

```
kconst = atof(params["K"].c_str());
```

というようなプログラムを書く必要がある。

params の key、data の組み合わせの UDF からの入力方法に関しては後述するが、プログラムにおいては複数の key、data の組み合わせからなる params を引数として取り、コンストラクタにおいて定義した private メンバーにデータを代入する。

B.3.4 calcForce の引数および内容

実際のポテンシャルの計算は、

```
double calcForce(const Vector3d& dr, Vector3d& ftmp);
```

というメソッドで行われるので、新たに定義したいポテシンャル関数をここに書く。Bond potential においては **dr**、**ftmp** を引数として取り、戻り値に計算したエネルギーを返すように中身を作成する。

引数 **dr** には結合ベクトル $\mathbf{r}_1 - \mathbf{r}_0$ が **double x,y,z** をメンバーに持つ class 構造で渡される。この **dr** とコンストラクタで値をセットした private メンバーを用いて、例に記すようにポテンシャルエネルギー U および力（$dU/d\mathbf{r}$）を計算する。計算した U は double 型として戻り値に与える。$dU/d\mathbf{r}$ は **ftmp** に代入することにより、本プログラムに値を渡すことができる。**ftmp** も **dr** と同様に **double x,y,z** のメンバーを持つクラスである。

Bond potential 以外の **calcForce** の引数は、ポテンシャルによって異なるので以下に示す。戻り値はすべて共通でポテンシャルエネルギーを返す

```
double UserAngle[1-3]::calcForce(const Vector3d dr[],Vector3d ftmp[]);
```

- **dr**[] ··· 結合角を構成する結合ベクトルの配列
  $\mathbf{r}_0 - \mathbf{r}_1 - \mathbf{r}_2$ で結合角が構成される場合、$\mathbf{dr}[0] = \mathbf{r}_1 - \mathbf{r}_0, \mathbf{dr}[1] = \mathbf{r}_2 - \mathbf{r}_1$ に値が渡される。
- **ftmp**[] ··· 計算した $dU/d\mathbf{r}_0$,$dU/d\mathbf{r}_1$,$dU/d\mathbf{r}_2$ の配列を返す

```
double UserTorsion[1-3]::calcForce(const Vector3d dr[],Vector3d ftmp[]);
```

- **dr**[] ··· 結合二面角を構成する結合ベクトルの配列
  $\mathbf{r}_0 - \mathbf{r}_1 - \mathbf{r}_2 - \mathbf{r}_3$ で結合二面角が構成される場合、$\mathbf{dr}[0] = \mathbf{r}_1 - \mathbf{r}_0, \mathbf{dr}[1] = \mathbf{r}_2 - \mathbf{r}_1, \mathbf{dr}[2] = \mathbf{r}_3 - \mathbf{r}_2$ に値が渡される。
- **ftmp**[] ··· 計算した $dU/d\mathbf{r}_0$,$dU/d\mathbf{r}_1$,$dU/d\mathbf{r}_2$,$dU/d\mathbf{r}_3$ の配列を返す

```
double UserPairInteraction[1-3]::calcForce(const Vector3d r[], Vector3d ftmp[]);
```

- **r**[] ··· Interaction site を定義する atom の座標の配列
  1 原子で定義する site 同士の相互作用の場合 **r[0]** に atom0 の座標、**r[1]** に atom1 の座標が入って渡される。2 原子で定義する site 同士の場合は **r[0]**、**r[1]** に site0 を定義する 2 つの atom の座標、**r[2]**、**r[3]** には site1 を定義する 2 つの atom の座標が入って渡される。
- **ftmp**[] ··· 計算した $dU/d\mathbf{r}_i$ を、**r**[] で渡された atom の Index に対応して、同じ長さの配列として返す。

```
double UserExternalField[1-3]::calcForce(
        const Vector3d r[], Vector3d ftmp[], SymMat3x3 vtmp[],Vector3d wtmp[]);
```

- **r**[] ··· Interaction site を定義する atom の座標の配列
- **ftmp**[] ··· **r**[] で渡された atom に対応した $dU/d\mathbf{r}_i$ を返す
- **vtmp**[] ··· 圧力を計算する際に用いるビリアル項 $-\mathbf{r}_i\mathbf{f}_j$ を **r**[] で渡された atom に対応した配列として返す。**vtmp**[] の各要素は **double xx, yy, zz, yz, zx, xy** をメンバーに持つクラスである。
- **wtmp**[] ··· External field として壁面のポテンシャルを与えて、壁面にかかる圧力 Wall pressure を計算する場合、**wtmp**[] に **r**[] で渡された atom に対応した force を含む配列を返す。
  たとえば、xy 平面上の z=0 and Zmax の位置に壁がある場合、**r[i]** に対応する **wtmp[i]** は
  $1/2(dU_{wall\,at\,z=0}(\mathbf{r})/d\mathbf{r} - dU_{wall\,at\,z=Zmax}(\mathbf{r})/d\mathbf{r})$ と定義する。
  （壁の面積で割ることにより圧力を計算するため、両面に壁がある場合、**COGNAC** における圧力の定義に従い、z=Zmax における force の符号を変えて和を取り平均する）定義する外場が壁面のポテンシャルでない場合、あるいは壁にかかる圧力を計算する必要がない場合は無視してよい。

B.3.5 UserPairInteraction[1-3]::calcDragForce の引数および内容

UserPairInteraction においては **calcForce** に加えて、DPD 法で計算される遥動－散逸力のように、2 原子間の距離および速度差より力を計算する関数、**calcDragForce** を定義することが出来る。
この関数の引数は以下のようになっている。

```
double UserPairInteraction[1-3]::calcForce(const Vector3d r[], const Vector3d v[],
                                           Vector3d fdissipative[], Vector3d frandom[])
```

- **r**[] ··· Interaction site を定義する atom の座標の配列
  1 原子で定義する site 同士の相互作用の場合 **r**[**0**] に atom0 の座標、**r**[**1**] に atom1 の座標が入って渡される。2 原子で定義する site 同士の場合は **r**[**0**]、**r**[**1**] に site0 を定義する 2 つの atom の座標、**r**[**2**]、**r**[**3**] には site1 を定義する 2 つの atom の座標が入って渡される。
- **v**[] ··· Interaction site を定義する atom の速度の配列
  配列のインデックスと atom との対応は **r**[] と同様
- **fdissipative**[] ··· 散逸項由来の力を、**r**[] で渡された atom の Index に対応して、同じ長さの配列として返す。
- **frandom**[] ··· ランダム力を、**r**[] で渡された atom の Index に対応して、同じ長さの配列として返す。
  圧力計算のためのビリアル項を計算する際、**fdissiaptive**[] は含まれるが **frandom**[] は含まれない。
- 戻り値　··· ポテンシャルエネルギー（通常用いることはない）

B.3.6 UserPairInteraction[1-3]::calcTailCorrection の引数および内容

UserPairInteraction においては **calcForce**、**calcDragForce** に加えて、ポテンシャルおよびビリアル項の tail correction を行うためのメソッド **calcTailCorrection** を追加することができる。以下に “src” ディレクトリにある “userpaitinteraction1.cpp” の中の **calcTailCorrection** の部分を例に示す。
この例では文献 [56] にしめされている方法で、Lennard-Jones ポテンシャルの tail correction を行っている。

```
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;
}
```

引数、戻り値は以下の内容である。

- **ene** ··· tail correction で補正されるポテンシャルエネルギー値
- **n1,n2** ··· システムに含まれる相互作用を計算する interaction site の数
  同一の site 同士の場合は n1=n2
- **volume** ··· ユニットセルの volume
- 戻り値 ··· tail correction で補正されるビリアル項
  SymMat3x3 という **double xx, yy, zz, yz, zx, xy** をメンバーに持つ class object（例では tailVirial）を作成し値を返す。

また上記例で使われているメンバーとして以下のものがある

- **cutOff** ··· cut off 距離
  PairInterction の基底クラスのメンバーとして持つ
- **PI** ··· $\pi = 3.1415....$ の値
  COGNAC のグローバルバラメータとして定義されている
- **sigmaSq**、**epsilon** ··· **UserPairInteraction1** の private メンバー

B.3.7 UserExternalField[1-3]::setCell の引数および内容

UserExternalField においては **calcForce** に加えて、ユニットセルの情報を引数として毎ステップ取り込んで、ユニットセルの変化に対応した、ポテンシャルを設定することが出来る。たとえば、体積変化に応じた外場の変化、壁のポテンシャルを設定する場合、壁表面積の変化によるポテンシャルの変化などを記述することが出来る。

以下に示す例は、**COGNAC** 組み込みの **ExternalField**、**LJWall** の例で、非直交セルになった場合、エラーを投げて終了するような処理を行っている。

```
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;
}
```

ここで用いられている引数 **cell** は **double xx, yy, zz, yz, zx, xy** をメンバーに持つクラスで、ユニットセルサイズ $a, b, c, alpha, beta, gamma$ が渡される（角度は degree）

また上記例で使われている **c_shape** は **ExternalField** クラスの private メンバーであり、**cell** と同じ **SymMat3x3** 型である。

B.4 COGNAC のリコンパイル

新たにユーザー定義ポテンシャルを追加した場合、**COGNAC** をリコンパイルする必要がある。一般的なコンパイルの方法は付録 A を参照。

その際に、たとえば “userbond2.cpp” のように実装ファイルを新たに作成した場合は、“Makefile” あるいは Visual studio の場合はワークスペースにソースファイルを追加する必要があるので注意。

B.5 UDF におけるパラメータ入力

実際に入力 UDF において、追加したユーザー定義ポテンシャル関数を使用する際の方法をしめす。例えば **User_Bond** の場合 §5.2.4 に示すように、**Index**、**Parameters[].Name**、**Parameters[].Value** に各々以下のようにデータを入力することにより、**User_Bond** として新たに作成したクラスにデータを渡すことが出来る。

- **Index** ··· User potential のクラス名の Index に対応する
  例えば **User_Bond** において **Index=1** と指定すると **UserBond1** クラスの内容がそのポテンシャルとして使用される。そのためこの Index は 1-3 の整数を取る。
- **Parameter[].Name** ··· コンストラクタの引数であるマップ、**params** の Key に対応する
  params のデータを指定する Key として任意の string data を与える
- **Parameter[].Value** ··· コンストラクタの引数であるマップ、**params** のデータに対応する
  入力においては、すべて string data として与えられるので前出のように必要に応じてコンストラクタで型変換しなければいけない。

Parameter[] は使用するユーザー定義ポテンシャル関数で使用する private 変数の数だけの要素を持つ必要がある。

付 録C　分子構造ファイルのインポート

COGNAC では、**SILK** というビルダーにより分子のアーキテクチャーを定義する方法に加え、市販（あるいはフリー）の分子ビルダーにより作成した、分子アーキテクチャーおよび立体構造を、**GOURMET** の **MoleculeBuilder** を用いて **COGNAC** 入力 UDF 上にインポートすることが出来る。ここでは、分子構造を記述する良く知られたフォーマットである mol ファイルを例にとり、ユナイテッドアトムモデルのテトラデカン（$C_{14}H_{30}$）1 分子の mol ファイルを **COGNAC** の UDF にインポートする。方法について、簡単なチュートリアルとして紹介する。

- 本チュートリアルでは、分子の結合情報と座標情報についてのファイル変換を　行うことのみを目的としいる
- Linux 環境で作業する場合は、ディレクトリ名を適当に読み替えること

1. **COGNAC** インプットファイルの作成

   作業用ディレクトリをつくる。（ここでは、“C:\work_octa” とする）“sample\molfile_sample” 以下にある “c14.mol” を、このディレクトリに置く。　次に、エディタで”test.udf” という名前のテキストファイルを　作成する。

   そのテキストファイルの一行目に

   ```
   \include {"cognac92.udf"}
   ```

   とだけ、記述する。保存先として” c:\work_octa\test.udf” を指定し、保存して終了する。（この段階では”c:\work_octa\test.udf” というファイルが作られている）

2. **GOURMET** から”test.udf”を開く

   GOURMET を起動し、”test.udf”を開く。（**GOURMET** のメニューから **File**→ **Open**）

3. ファイルのインポート

   GOURMET のメニューから　**Tool** → **MoleculeBuilder**　を選択する。　**MoleculeBuilder** の画面がポップアップされるので以下の情報を入力する。

   ```
   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** とは、変換元のファイルを指す。
- 本例では、Input UDF と Output UDF を同じ名前”test.udf”にすること
- **GenerateElectrostatic Site** のチェックボックスは **OFF**（チェックをつけない）にすること。

次に、**OK** を選択する　**Disregard Current Changing?**　いうメッセージに対して **Yes** を選択。

すると、Open している”test.udf” の **Set_of_Molecules** と **Structure** に、新しく分子のアーキテクチャーおよび座標データがインポートされる。

4. 変換結果の描画

GOURMET 上で、**VIEWER_show** を用いて描画する。（第 3 章中 **Action による分子構造の表示**の項参照）

- 立ち上がるウィンドウに示される選択項目の内、**bc** を **off** とすること。mol file をインポートした時点では境界条件が設定されていないので、**bc** = **on** あるいは **atom** にすると分子構造が正しく表示されない。

テトラデカン（ 1 分子）のユナイテッドアトムモデル（トランス構造）が　線画で表示される。

参考文献

1) Aoyagi, T., Sawa, F., Shoji, T., Fukunaga, H., Takimoto, J. and Doi, M.: *Comput. Phys. Comm.*, Vol. 145, p. 267 (2002).

2) （公社）新化学技術推進協会（編）: 高分子材料シミュレーション—OCTA 活用事例集—, 化学工業日報社 (2014).

3) （公社）新化学技術推進協会（編）: 高分子材料シミュレーション—OCTA 活用事例集—増補版, 化学工業日報社 (2017).

4) Allen, M. P. and Tildesley, D. J.: *Computer Simulation of Liquids*, Clarendon Press, Oxford (1987).

5) 岡田勲, 大澤映二（編）: 分子シミュレーション入門, 海文堂 (1989).

6) D.W. ヘールマン著, 小澤　哲, 篠島　妥訳 : シミュレーション物理学, シュプリンガー・フェアラーク東京 (1990).

7) 神山新一, 佐藤明 : 分子シミュレーション講座 2 分子動力学シミュレーション, 朝倉出版 (1997).

8) 川添良幸, 三上益弘, 大野かおる : コンピュータ・シミュレーションによる物質科学, 共立出版 (1996).

9) Willian G.Hoover 著, 田中　實訳 : 分子動力学入門, 共立出版 (1998).

10) 佐藤明 : HOW TO 分子シミュレーション, 共立出版 (2004).

11) 岡崎進, 吉井範行 : コンピュータ・シミュレーションの基礎　第 2 版, 化学同人 (2011).

12) Groot, R. D. and Warren, P. B.: *J. Chem. Phys.*, Vol. 107, p. 4423 (1997).

13) Groot, R. D. and Madden, T. J.: *J. Chem. Phys.*, Vol. 108, p. 8713 (1998).

14) Rigby, D. and Roe, R. J.: *J. Chem. Phys.*, Vol. 87, p. 7285 (1987).

15) Hoover, W. G.: *Phys. Rev. A*, Vol. 31, p. 1695 (1985).

16) Berendsen, H. J. C., Postma, J. P. M., Gunsteren, van W. F., Dinola, A. and Haak, J. R.: *J. Chem. Phys.*, Vol. 81, p. 3684 (1984).

17) Grest, G. S. and Kremer, K.: *Phys. Rev. A*, Vol. 33, p. 3628 (1986).

18) Andersen, H. C.: *J. Chem. Phys.*, Vol. 72, p. 2384 (1980).

19) Parrinello, M. and Rahman, A.: *J. Appl. Phys.*, Vol. 52, p. 7182 (1981).

20) Brown, D. and Clarke, J. H. R.: *Macromolecules*, Vol. 24, p. 2075 (1991).

21) Lees, A. W. and Edwards, S. F.: *J. Phys. C:Solid State Phys.*, Vol. 5, p. 1921 (1972).

22) Tuckerman, M. E., Mundy, C. J. and Balasubramanian, S.: *J. Chem. Phys.*, Vol. 106, No. 13, p. 5615 (1997).

23) Allen, M. P. and Tildesley, D. J.: *Computer Simulation of Liquids*, chapter 3, p. 73, Clarendon Press, Oxford (1987).

24) Andersen, H. C.: *J. Comput. Phys.*, Vol. 52, p. 24 (1983).

25) Kremer, K. and Grest, G. S.: *J. Chem. Phys.*, Vol. 92, p. 5057 (1990).

26) *Amber 8 User's Manual (http://amber.scripps.edu/)*.

27) Mayo, S. L., Olafson, B. D. and Goddard, W. A.: *J. Phys. Chem.*, Vol. 94, p. 8897 (1990).

28) Bedrov, D., Smith, G. D. and Smith, J. S.: *J. Chem. Phys.*, Vol. 119, p. 10438 (2003).

29) Gay, J. G. and Berne, B. J.: *J. Chem. Phys.*, Vol. 74, p. 3316 (1981).

30) Cleaver, D. J., Care, C. M., Allen, M. P. and Neal, M. P.: *Phys. Rev. E*, Vol. 54, p. 559 (1996).

31) Grest, G. S.: *J. Chem. Phys.*, Vol. 105, p. 5532 (1996).

32) Aoyagi, T., Takimoto, J. and Doi, M.: in *MRS 2000 Spring Meeting, Abstracts*, p. 260 (2000).

33) 青柳岳司, 滝本淳一, 土井正男： 高分子学会予稿集, 第 49 巻, p. 443 (2000).

34) 平成 12 年度　高機能材料設計プラットフォームの研究開発　(エネルギー使用合理化技術開発) 成果報告書, Technical report, (財) 化学技術戦略推進機構 (2001).

35) Evans, D. J. and Morriss, G. P.: *Statistical Mechanics of Nonequilibrium Liquids*, Academic press (1990).

36) Onsager, L.: *J. Am. Chem. Soc.*, Vol. 58, p. 1486 (1936).

37) Barker, J. A. and Watts, R. O.: *Mol. Phys.*, Vol. 26, p. 789 (1973).

38) Nymand, T. M. and Kinse, P.: *J. Chem. Phys.*, Vol. 112, p. 6152 (2000).

39) Groot, R. D.: *J. Chem. Phys.*, Vol. 118, p. 11265 (2003).

40) Eastwood, J. W., Hockney, R. W. and Lawrence, D. N.: *Comput. Phys. Commun.*, Vol. 19, p. 215 (1980).

41) Beckers, J. V. L., Lowe, C. P. and Leeuw, S. W. D.: *Molecular Simulation*, Vol. 20, p. 369 (1998).

42) 青柳岳司, 滝本淳一, 土井正男： 高分子学会予稿集, 第 49 巻, pp. 2569–2570 (2000).

43) Allen, M. P. and Tildesley, D. J.: *Computer Simulation of Liquids*, chapter 8, p. 242, Clarendon Press, Oxford (1987).

44) 青柳岳司, 滝本淳, 土井正男： 第 14 回分子シミュレーション討論会講演要旨集, pp. 141–142 (2001).

45) Magatti, D. and Ferri, F.: *Applied Optics*, Vol. 40, No. 24, p. 4011 (2001).

46) Fukunaga, H., Takimoto, J., Aoyagi, T., Shoji, T., Sawa, F. and Doi, M.: *Prog. of Theoret. Phys. Suppl.*, Vol. 138, p. 396 (2000).

47) Fukunaga, H., Takimoto, J. and Doi, M.: *Mol. Cryst. Liquid Cryst.*, Vol. 365, p. 739 (2001).

48) Aoyagi, T., Honda, T. and Doi, M.: *J. Chem. Phys.*, Vol. 117, No. 17, p. 8153 (2002).

49) Aoyagi, T., Takimoto, J. and Doi, M.: *J. Chem. Phys.*, Vol. 115, p. 552 (2001).

50) Bunn, C. W.: *Trans. Faraday Soc.*, Vol. 35, p. 482 (1939).

51) Nose, S.: *Molecular Physics*, Vol. 52, p. 255 (1984).

52) Fukuda, M. and Kuwajima, S.: *J. Chem. Phys.*, Vol. 107, No. 6, p. 2149 (1997).

53) Milano, G. and Kawakatsu, T.: *J. Chem. Phys.*, Vol. 130, No. 21, p. 214106 (2009).

54) Jorgensen, W. L., et al.: *J. Comput. Chem.*, Vol. 11, p. 958 (1990).

55) Chandrasekhar, J., et al.: *J. Am. Chem. Soc.*, Vol. 106, (1984).

56) Allen, M. P. and Tildesley, D. J.: *Computer Simulation of Liquids*, chapter 2, p. 64, Oxford (1987).