OCTA

ソフトマテリアルのための統合化シミュレータ

多相構造シミュレータ

Muffin

version 5.1

ユーザーズマニュアル

- 第1分冊 -

共通機能解説

OCTA ユーザーズグループ

January 01 2016

執筆者

Introduction	山上達也
Muffin	山上達也、谷口貴志
Milk and Tools	山上達也、佐々木誠
Appendix	山上達也

プログラム開発者

Muffin	山上達也、谷口貴志
Milk	山上達也、黒田明義
Python Tools	山上達也、佐々木誠

バージョン 3.3 リリース

プログラム開発、マニュアル執筆　山上達也

バージョン 4.0 リリース

プログラム開発、マニュアル執筆　山上達也

バージョン 4.1 リリース

プログラム開発、マニュアル執筆　山上達也

バージョン 5.0 リリース

プログラム開発、マニュアル執筆　山上達也、小沢拓

バージョン 5.1 リリース

プログラム開発、マニュアル執筆　小沢拓

謝辞

本プログラム開発の初版は、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 (OCTA プロジェクト)」の下で行われたものである。

また、2003 年度からの本プログラム開発の一部は、科学技術振興機構 (JST)・戦略的創造研究推進事業 (CREST) ・バイオレオプロジェクト (2002 年度採択事業) の支援の下で行われたものである。

目 次

図 目 次

表 目 次

はじめに

本マニュアルは多相構造ダイナミクス

Multi-Phase Dynamics

を取り扱うシミュレータである

Muffin

(MUltiFarious Field Simulator for Non-equilibrium system)

(以下、***Muffin***) の機能と入出力データに関して解説したものである。

Muffin は、温度や圧力などの熱力学変数の変化、電場や応力の印可など外場の変化による高分子流体の流れや高分子固体の変形とそれに伴う多相構造間の相転移や相構造形成ダイナミクスを取り扱うシミュレータである。本リリースでは Muffin を、以下の 6 つのパッケージ (3 つの流体系パッケージ、2 つの固体系パッケージ、1 つの光学系パッケージ) に分けてリリースする。

1. PhaseSeparation. ... 多相流体シミュレータ (FDM および FEM).
2. Electrolyte. ... 電解質流体シミュレータ (FDM および FEM).
3. MEMFluid. ... マイクロ電気化学チップ流体シミュレータ (FEM).
4. Elastica. ... 線形弾性体シミュレータ.
5. Geldyn. ... ゲル大変形動力学シミュレータ.
6. Turban. ... 光透過性シミュレータ.

本マニュアルは多相構造シミュレータ Muffin マニュアル第 I 分冊として、Muffin の起動方法と入出力 UDF、ダイナミクスマネージャなどの基本機能、補助ツールについて解説する。各々のパッケージの機能と使用方法、応用例は II-VII の各分冊で詳細に解説する。第 II 分冊では、多相流体シミュレータ PhaseSeparation パッケージの有限差分法 (FDM) の PhaseSeparation_FDM と有限要素法 (FEM) の PhaseSeparation_FEM について、第 III 分冊では、電解質流体シミュレータ Electrolyte パッケージの有限差分法 (FDM) の Electrolyte_FDM と有限要素法 (FEM) の Electrolyte_FEM について、第 IV 分冊では、マイクロ電気化学チップ流体シミュレータ MEMFluid パッケージについて、第 V 分冊では、線形弾性体シミュレータ Elastica について、第 VI 分冊では、ゲル大変形動力学シミュレータ Geldyn について、第 VII 分冊では、光透過性シミュレータ Turban について、詳細に解説する。

補助ツールは大きく分けて、プリプロセッサ（メッシュ生成プログラム Milk、Elastica および Milk 用の入力支援モデラー、NASTRAN 形式の有限要素データの変換プログラム、場の初期値生成プログラム）、ポストプロセッサ（解析プログラム、描画プログラム、Plot プログラム）の 2 つからなる。付録 A は、Muffin のモジュールを再構築するためのコンパイル方法の解説。付録 B は、Muffin の骨格をなす場やメッシュ、ダイナミクスマネージャなどの C++クラスライブラリのソースコードと、Muffin に新しい機能を追加する手順の簡単な解説である。

我々の提供する Muffin 基本クラスライブラリを用いれば、非構造格子、構造格子、有限要素法 (FEM)、有限差分法 (FDM) などさまざまなシミュレータが容易に構築でき GOURMET との入出力も自動的に行うことができる。また、Muffin の用意する共通 UDF 解析プログラムも利用できる。

最後に、本シミュレータが高分子科学や Soft Matter Physics の研究において全く新しい意味ある結果を引き出すツールになることを願う。また、このシミュレータがソフトウェア工学的な意味においても斬新なものであり、今後、更なる発展を遂げることを願う。

2002 年 2 月 22 日 Muffin 開発者

バージョン 3.3 リリースに当たって

バージョン 3.2 のリリースからほぼ 1 年に、ユーザーの方々に頂いた幾つかのご意見や不具合を元に、少しですがプログラムやマニュアルの修正、サンプルの追加を行いました。マイナーバージョンアップ版をリリース出来ることを喜ばしく思います。これから益々、Muffin ユーザの方が増えることを祈って、メリークリスマス。

2002 年 12 月 25 日 Muffin 開発者

バージョン 4.0 リリースに当たって

バージョン 4.0 リリースでは、実験の写真など画像ファイルの読み込み機能、多孔質構造作成プログラム、アクションメニューの見直し、マニュアルの分冊化などを行いました。少し使い易くなっていれば幸いです。Muffin ユーザが益々増えることを祈って、メリークリスマス。

2003 年 12 月 25 日 Muffin 開発者

バージョン 4.1 リリースに当たって

バージョン 4.1 リリースでは、Milk と Elastica を中心に機能追加やモデラーの構築、action からのエンジンの簡易実行、チュートリアルの整備、サンプルの充実を行いました。FEM 系のソルバーの使い勝手が大幅に向上しているのではないかと思います。御利用下さい。Muffin の益々の発展を祈って !!

2005 年 3 月 3 日 Muffin 開発者

バージョン 5.0 リリースに当たって

バージョン 5.0 リリースでは、FDM 系のソルバー (PhaseSeparation、Electrolyte) が統合化されています（2006 年 9 月）。PhaseSeparation(FDM) への機能追加、PhaseSeparation、Elastica のマニュアルの一部改訂を実施しました（他のエンジンのマニュアルは整備中です）。MUFFIN を御利用下さい。

2015 年 2 月 1 日 Muffin 開発者

バージョン 5.1 リリースに当たって

バージョン 5.1 リリースでは、PhaseSeparation(FDM) への機能追加を行いました。相分離構造上での熱や物質の拡散計算が可能になりました。また、PhaseSeparation(FDM) と Elastica(FEM) の描画 Action を改良しました。MUFFIN を御利用下さい。

2016 年 1 月 1 日 Muffin 開発者

第1章　Muffinとは

Muffin はソフトマテリアルの多相構造を有限要素法 (FEM) および有限差分法 (FDM) で解析する汎用連続場シミュレータである。Muffin は次のパッケージ、**PhaseSeparation**, **Electrolyte**, **MEMFluid**, **Elastica**, **Geldyn**, **Turban** を含んでおり、複合材料の弾性挙動、ゲルの膨潤と収縮、電解質流体の界面動電現象、細管内部での化学反応と拡散、せん断や電場下での相分離と液滴の変形、球晶構造と光透過性の予測、などのソフトマテリアルに関連するさまざまな問題を扱う。また、Muffin は SUSHI で得られた相分離構造を元に、平均弾性率などのさまざまな平均物性を解析できる。これらの様々な機能を有機的に結合し解析を進め、将来の機能拡張性も高めるために、Muffin はオブジェクト指向設計により、場とソルバー、ダイナミクスを明確に機能分離し、機能間のインターフェースを統一し、さまざまなソルバーを持つ場のオブジェクトを自在に組み合わせ連携し解析する機能 (***DynamicsManager***) を実現している。

Muffin の備える解析機能と計算手法の一覧を示す。

1. Muffin の基本解析機能一覧
   - せん断場など、さまざまな流動場下での流動挙動の解析。
   - 液滴の合体分裂過程と多成分相分離構造の解析。
   - 電解質の電場下での流動挙動の解析。
   - 流体の化学反応と拡散流動および反応生成物の分離・抽出。
   - 応力印可、歪み印可での多相構造を持つ線形弾性体の変形解析。
   - ゲルおよび弾性体の応力・圧力・電場印加、温度・溶媒変化に伴う大変形。
   - 球晶構造の予測と光透過性の予測。
2. Muffin の基本計算手法一覧
   - 構造格子と有限差分法 (FDM)、Euler 描像による流動挙動解析。
   - 非構造格子と有限要素法 (FEM)、Euler 描像により任意形状での流動挙動解析。
   - 非構造格子と有限要素法 (FEM)、Lagrange 描像により任意形状でのゲル・弾性体変形解析。
   - 様々な場、ソルバ、ダイナミクスを連携して解くための Dynamics Manager。
   - デローネ自動分割による非構造格子の生成。
   - SUSHI とのズーミングを容易にする、メッシュや場、境界条件の共通 UDF の使用。

本リリースでは、これらの機能を以下のシミュレータに分けてリリースする。FDM 系のシミュレータは 1 つのパッケージに統合されている。

1. Multi-Fluid Phase Dynamics Simulator : PhaseSeparation ... 多相流体シミュレーションパッケージ
   (a) FDM PhaseSeparation Simulator (muffin5) ... FDM 多相流体シミュレータ
   (b) FEM PhaseSeparation Simulator (muffin5e_phaseseparation) ... FEM 多相流体シミュレータ
2. Electrolyte Fluid Dynamics Simulator : Electrolyte ... 電解質流体シミュレーションパッケージ

   (a) FDM Electrolyte Simulator (muffin5) . . . FDM 電解質流体シミュレータ
   (b) FEM Electrolyte Simulator (muffin5e_electrolyte) . . . FEM 電解質流体シミュレータ

3. Micro Electro Mechanical Fluid Dynamics Simulator : MEMFluid (muffin5e_memfluid) . . . マイクロ電気化学チップ流体シミュレータ
4. Linear Elasticity Simulator : Elastica (muffin5e_elastica) . . . 多相線形弾性体シミュレータ
5. Gel Dynamics Simulator : GelDyn (muffin5e_geldyn) . . . ゲル大変形動力学シミュレータ
6. Light Transmittance Simulator : TURBAN (turban) . . . 光透過性シミュレータ

これらのモジュールのうち、流体系と固体系のモジュールは、各々、シミュレーション対象や機能は異なるが、Muffin 共通の機能である、場・メッシュ・境界条件・ダイナミクスマネージャ機能とこれらの入出力機能の上に構築されており、操作方法や入出力 UDF ファイル、解析プログラムは統一されている。

第2章　起動方法

ここでは、インストール解説書に沿ってインストールを行った直後の状態を想定し、Muffin の起動方法の解説を行う。

2.1　起動方法

2.1.1　起動に必要な UDF ファイル

Muffin を使用するには、以下の UDF ファイルが必要となる。

- UDF 定義ファイル (muffin5.udf または muffin5e.udf) . . . 入出力 UDF のデータ構造が定義されている。このファイルは環境変数 UDF_DEF_PATH で指定されたディレクトリに置く必要がある。インストールマニュアルに従いエンジンをインストールした状態で、既に設定されているが、手動で設定する場合には、エンジンをインストールしたディレクトリ (環境変数 PF_ENGINE) の下の udf ディレクトリ内にファイルがあるので、このディレクトリに環境変数 UDF_DEF_PATH を設定する必要がある。UDF_DEF_PATH を定義しない場合は、エンジンを起動したディレクトリに UDF 定義ファイルが存在する必要がある。
- 入力 UDF ファイル (inputUDF) . . . 入力データが記述された UDF
- 出力 UDF ファイル (outputUDF) . . . 出力データが記述される UDF

また、オプションでエンジンのリアルタイム結果表示・制御を行なうには、以下のファイルが必要となる。

- 解析結果出力 UDF 定義ファイル (muffin5res.udf) . . . 解析結果出力 UDF のデータ構造が定義されている。
  このファイルは同様に環境変数 UDF_DEF_PATH で指定されたディレクトリに置く必要がある。
- コントロールパラメータ入力 UDF ファイル (parameterUDF) . . . 計算途中で変更するパラメータが記述された入力 UDF
  muffin5par.udf の UDF 定義ファイルを用いて作成する。
- 解析結果出力 UDF ファイル (summaryUDF) . . . 解析結果出力データが記述される UDF

2.1.2　起動コマンド

本リリースでの実行モジュール名は、各々、以下の通りである。

- FDM 多相流体シミュレータ . . . muffin5
- FEM 多相流体シミュレータ . . . muffin5e_phaseseparation
- FDM 電解質流体シミュレータ . . . muffin5
- FEM 電解質流体シミュレータ . . . muffin5e_electrolyte

- マイクロ電気化学反応流体シミュレータ ... muffin5e_memfluid
- 多相線形弾性体シミュレータ ... muffin5e_elastica
- ゲルダイナミクスシミュレータ ... muffin5e_geldyn

引数や入出力 UDF ファイルの構造などモジュールで共通であるので、以下、モジュール名を muffin と書いて解説する。

プロンプトからの起動

unix 系/linux 系/cygwin ではシェルプロンプト、windows(win32、win64) ではコマンドプロンプトから、MUFFIN の上記の実行モジュールに引数を与えて実行する。

以下のコマンドラインにより Muffin を実行できる。{} で囲まれた引数はオプションである。主に-I と-O が用いられる。

% muffin(module name) -I *inputUDF* {-O *outputUDF*} {-r *or* -i *input_record_No.*} {-e *end_record_No.*} {-P *parameterUDF*} {-S *summaryUDF*}

- inputUDF ... 入力 UDF ファイルのパス
- outputUDF ... 出力 UDF ファイルのパス
- input_record_No. ... 入力する UDF データのレコード番号。-r は FDM エンジン、-i は FEM エンジンで使用。
- end_record_No. ... 解析を実行し、出力する最大計算ステップ数
- parameterUDF ... インタラクティブなパラメータ変更コマンドの入力 UDF ファイルのパス
- summaryUDF ... インタラクティブな計算結果監視のための解析結果出力 UDF ファイルのパス
- GOURMET からのエンジン実行では、-P で指定されたコントロールパラメータファイルが、初期化処理で 1 回必ず読まれ、その後は、(パラメータが変更され) エンジンが”RESUME” された時に読まれる。-S で指定された解析結果出力ファイルには、コントロールパラメータファイルで指定された解析結果データが、入力パラメータ “INTERVAL_OF_MONITORING” のステップ間隔で出力される。コントロールパラメータファイルに出力する解析結果データが指定されなかった場合には、すべての解析結果データが解析結果出力ファイルに出力される。
- エンジンをコンソールから直接実行するバッチ処理でも、-P や -S オプションを用いて、解析結果データを出力することができる。

  バッチ処理の場合、-P で指定されたコントロールパラメータが読まれるのは、初期化の際の一度だけである。-S で指定された解析結果出力ファイルには、コントロールパラメータファイルで指定された解析結果データが、入力パラメータ “INTERVAL_OF_MONITORING” のステップ間隔で出力される。コントロールパラメータファイル (-P) が指定されなかった場合や、コントロールパラメータファイルに出力する解析結果データが指定されなかった場合には、すべての解析結果データが解析結果出力ファイルに出力される。

第3章　リファレンス

3.1　入出力 UDF 解説

3.1.1　入出力 UDF 概要

Muffin で入力 UDF ファイルと出力 UDF ファイルの構造は同じである。以下に入出力 UDF ファイルの概要を解説する。

GOURMET の Editor で見たときの図を示す.

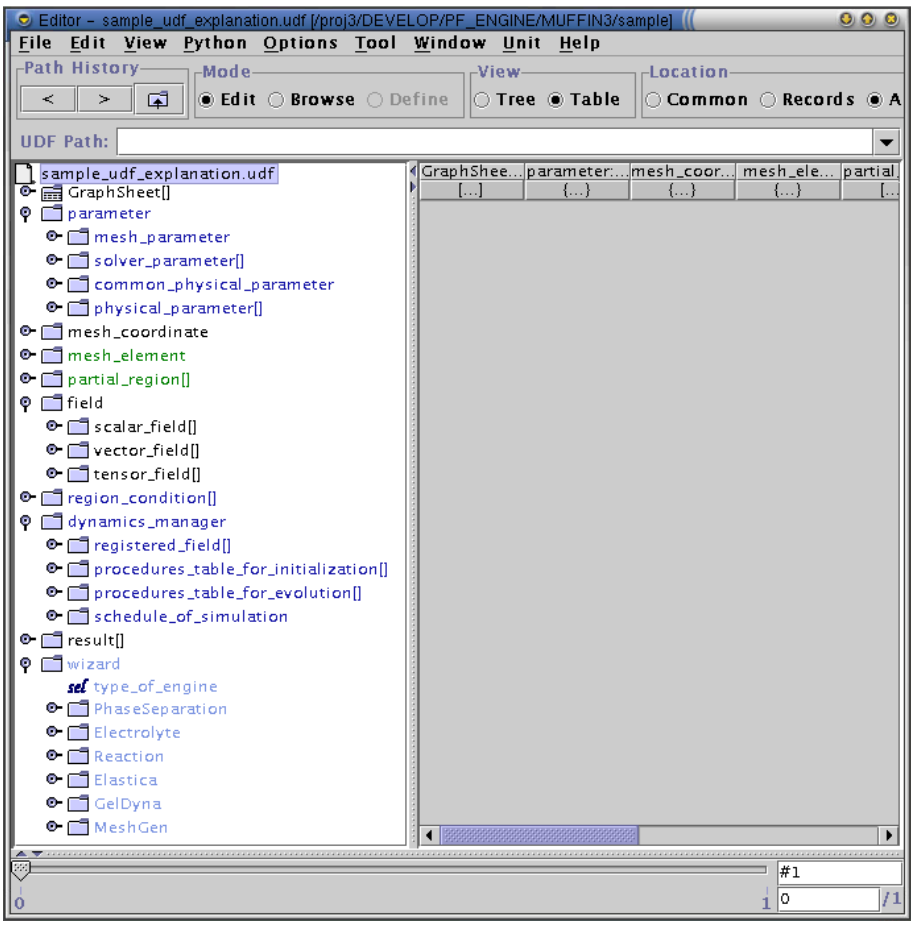


図 3.1: Muffin の入出力 UDF のトップオブジェクト (UDF 解説)

図 3.1 に示すように、**Muffin** の入出力データは以下の構造を持つ。

1. パラメータ部 (parameter)
   - メッシュ構造パラメータ (mesh_parameter)
   - ソルバ制御パラメータ (solver_parameter)

- 共通 物理パラメータ (common_physical_parameter)
- 物理パラメータ (physical_parameter)

2. メッシュ節点座標データ部 (mesh_coordinate)
3. メッシュ要素連結データ部 (mesh_element)
4. 部分領域データ部 (partial_region[])
5. 場のデータ部 (field)
   - スカラー場データ (scalar_field)
   - ベクトル場データ (vector_field)
   - テンソル場データ (tensor_field)
6. 部分領域 (境界) 条件データ部 (region_condition[])
7. ダイナミクスマネージャ部 (dynamics_manager)
   - 使用する場のリスト (registered_field)
   - 初期化手続きのリスト (procedures_table_for_initialization)
   - 1 ステップ実行手続きのリスト (procedures_table_for_evolution)
   - 実行するシミュレーションスケジュールのデータ (schedule_of_simulation)
8. 解析結果データ部 (result[])
9. モデラー部 (modeler)

大きく、パラメータ、メッシュ節点座標データ、メッシュ要素連結データ、部分領域データ、場のデータ、部分領域 (境界) 条件データ、ダイナミクスマネージャ、解析結果データ、モデラー部より構成される。

UDF データには、コモンデータ (レコードによらず共通なデータ) とレコードデータの区別があるが、各々のデータは以下のように分類されて、入出力される。

1. パラメータ部 (parameter) . . . 初期化データ
2. (FEM のみ) メッシュ節点座標データ部 (mesh_coordinate)
   - 流体系シミュレータ (オイラー描像) . . . 初期化データ
   - 固体系シミュレータ (ラグランジュ描像) . . . レコードデータ
3. (FEM のみ) メッシュ要素連結データ部 (mesh_element) . . . 初期化データ
4. (FEM のみ) 部分領域データ部 (partial_region[]) . . . 初期化データ
5. 場のデータ部 (field) . . . レコードデータ
   (但し、コモンデータに記述し入力すると場の初期値分布として読み込まれる)
6. 部分領域 (境界) 条件データ部 (region_condition[]) . . . 初期化データ
7. ダイナミクスマネージャ部 (dynamics_manager) . . . 初期化データ
8. 解析結果データ部 (result[]) . . . レコードデータ
9. モデラー部 (modeler) . . . 初期化データ

以下に、パラメータ、メッシュ節点座標データ、メッシュ要素連結データ、部分領域データ、場のデータ、部分領域 (境界) 条件データ、ダイナミクスマネージャ、解析結果データの各データを解説する。

本リリースでは、ユーザーがダイナミクスマネージャの編集をすること無く、一通りのことは出来るように、Muffin で計算可能な一通りのサンプル UDF をつけており、基本操作、応用操作で解説しているので、これらのサンプルのパラメータ値や境界条件データを編集して使用すれば十分と思われる。

新しいソルバを開発し追加する方など、ダイナミクスマネージャの詳細に興味のある方は、3.2 をご覧頂きたい。

3.1.2 パラメータ部 (parameter)

パラメータ部 (parameter) の概要

パラメータは、すべて入力 UDF のコモンデータより入力され、そのまま出力 UDF のコモンデータに出力される。パラメータは、大きく以下の 4 つに分けられている。

1. メッシュ構造パラメータ (mesh_parameter)
   メッシュ作成のためのパラメータ群の値を与える。
2. ソルバ制御パラメータ (solver_parameter)
   ソルバー・チューニング用パラメータを与える.
   (SOR 法加速因子,CG 反復解法における繰り返し最大回数, など)
3. 共通 物理パラメータ (common_physical_parameter)
   MUFFIN のずべてのモジュールに共通して使われるパラメータを与えるところである。例えば時間刻み DT, 初期ステップ数 INITIAL_STEP, 初期時刻 INITIAL_TIME, 終了ステップ数 FINAL_STEP がある。
4. 物理パラメータ (physical_parameter)
   シミュレーションで用いる物理定数、又は無次元化パラメータを与える.

メッシュ構造パラメータ (parameter.mesh_parameter)

Mesh の生成に関する全ての情報が書かれる所である。GOURMET の Editor 画面に沿って解説する。
図 3.2 に Editor で、mesh_parameter 部を展開した様子を示す。ここで、

- name : . . . メッシュにつける名前 (何でも良い)
  1 つのシミュレーションで複数のメッシュデータを用いる場合には意味を持つ (将来) が、現状は 1 つのシミュレーションで 1 メッシュである。
- type : . . . メッシュのタイプ
  メッシュのタイプとしては下記の 4 タイプの構造格子と 4 タイプの非構造格子をサポートしている。
  - REGULAR . . . 規則メッシュ(FDM のみ)
  - UNSTRUCTURED_RECT . . . 非構造メッシュ(四角形、直方体 形状)(FEM のみ)
  - UNSTRUCTURED_SPHERE . . . 非構造メッシュ(円、球 形状)(FEM のみ)
  - UNSTRUCTURED_INPUT . . . 非構造メッシュデータ入力 (リスタート用)(FEM のみ)

図 3.3 に Editor で、axes[].values[] を展開した様子を示す。ここで、

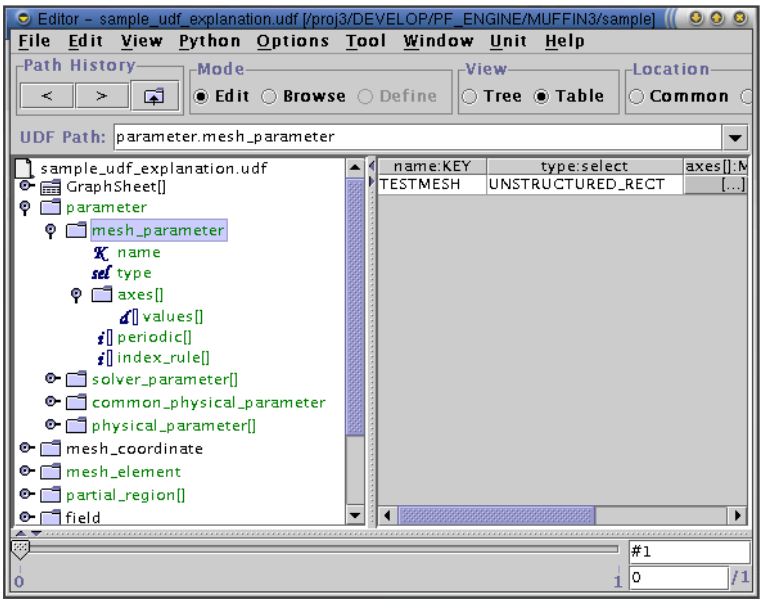


図 3.2: メッシュパラメータ:メッシュタイプの入力画面 (UDF 解説)

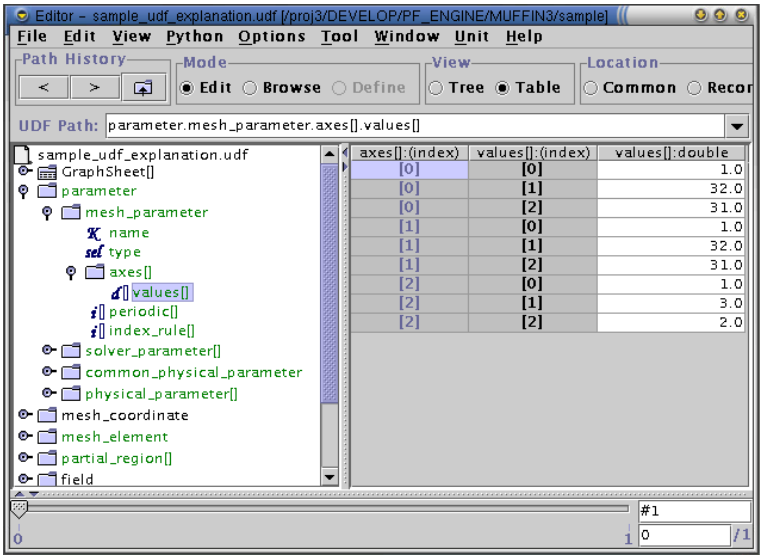


図 3.3: メッシュパラメータ:軸データの入力画面 (UDF 解説)

- axes[] : . . . 軸データの配列
  axes[] の配列の要素数が空間次元を表す。

  図 3.3 は 3 次元の場合であり、axes[0], axes[1], axes[2] が、各々、X, Y, Z 軸を表す。

- axes[].values[] : . . . 各々の軸の座標や分割数のデータ
  メッシュタイプが、REGULAR, UNSTRUCTURED_XXXX の場合

  – axes[].values[0] . . . 軸の領域の最小値
  – axes[].values[1] . . . 軸の領域の最大値
  – axes[].values[2] . . . 軸の分割数
    但し、非構造格子データ入力 (UNSTRUCTURED_INPUT) の場合には軸の分割数は無視される。

  図 3.3 は X, Y, Z 軸、何れも、最小値 1.0, 最大値 2.0, 分割数 15 であることを示す。

- unit : . . . 座標の単位

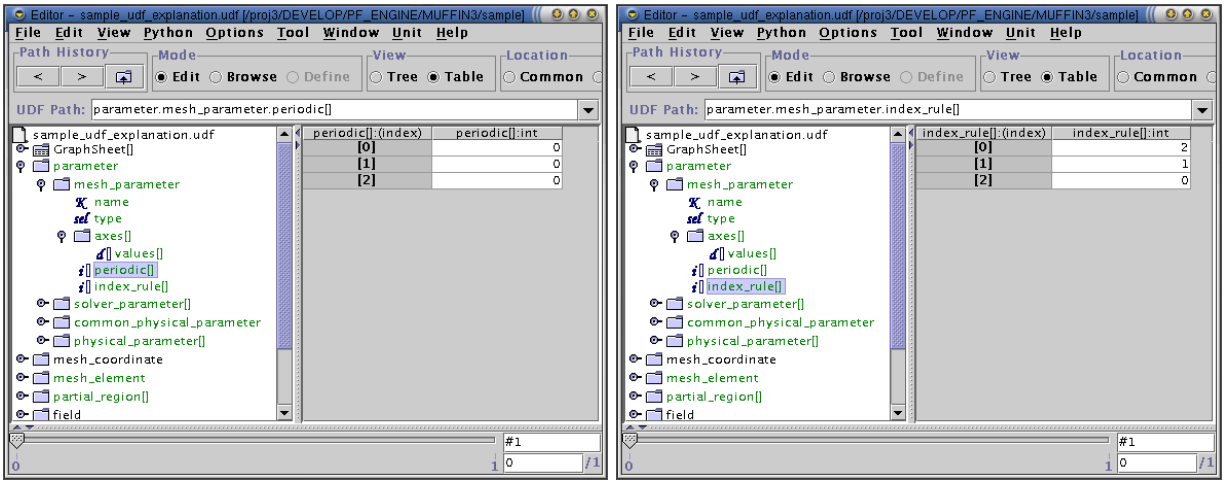


図 3.4: メッシュパラメータ:周期境界条件と INDEX_RULE の入力画面 (UDF 解説)

図 3.4 に Editor で、periodic[] および index_rule[] を展開した様子を示す。ここで、

- periodic[] : . . . 軸の周期境界条件データの配列
  1 . . . 周期、0 . . . 非周期を表す。periodic[0], periodic[1], periodic[2] は、各々、axes[0], axes[1], axes[2] の軸の周期性に対応する。

  図 3.4 は 3 次元の場合であり、X, Y, Z 軸すべて非周期であることを表す。

- index_rule[] : . . . 構造格子の場合、場のデータ配列の順序と軸の回り順序の関係を記述する。

[規約 01]

例えば、場のデータの並び順序とメッシュの軸の関係が、[(0,0,0),(0,0,1),..(0,0,Z-1),(0,1,0),(0,1,1),..(0,1,Z-1), ...(X-1,Y-1,0),(X-1,Y-1,1),..(X-1,Y-1,Z-1)] と、Z が一番早く変化し、Z,Y,X の順で早く変化している場合は index_rule[] = [2,1,0] (MUFFIN の構造格子はこれ) となる。
例えば、場のデータの並び順序とメッシュの軸の関係が、[(0,0,0),(1,0,0),..(X-1,0,0),(0,1,0),(1,1,0),..(X-1,1,0), ...(0,Y-1,Z-1),(0,Y-1,Z-1),..(X-1,Y-1,Z-1)] と、X が一番早く変化し、X,Y,Z の順で早く変化している場合は index_rule[] = [0,1,2] (SUSHI の構造格子はこれ) となる。

ソルバ制御パラメータ (parameter.solver_parameter)

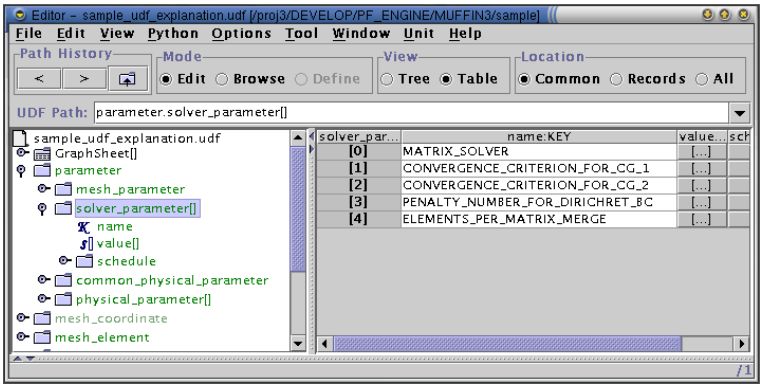


図 3.5: ソルバ制御パラメータ (UDF 解説)

ここには、収束の Criterion, 繰り返し法を解放として使用するの場合の最大の繰り返し回数, SOR ソルバーの加速因子の値 など、数値計算アルゴリズム上パラメータだけを書くことにする.

[規約 02]

最大繰り返し回数に関しては、 MAX_ITERATION_FOR_ という prefix を
収束判定に使うの定数には、 CONVERGENCE_CRITERION_FOR_という prefix を
加速因子に関しては ACCELERATION_VALUE_FOR_ という prefix を付けるようにする.

図 3.5 に、Elastica でのソルバ制御パラメータを例に示す。

共通 物理パラメータ (parameter.common_physical_parameter)

ここには、Muffin の全てのシミュレータに共通なパラメータを記述する。common_physical_parameter として、現在、以下の 6 つのパラメータが存在する.

- DT: . . . 時間方向に積分する為の時間刻み
  この値を設定すれば、ダイナミクスマネージャのタイムカウンターが 1STEP 実行毎に自動的に DT だけ足し込まれる。スケージュールのコントロールに時刻を用いる場合には必須である。

- INITIAL_STEP: . . . リスタートデータのレコード番号 (デフォルト値,0)
  ダイナミクスマネージャのタイムステップカウンターの初期値になる。
  (注意: INITIAL_STEP は 0 を含む正の整数)

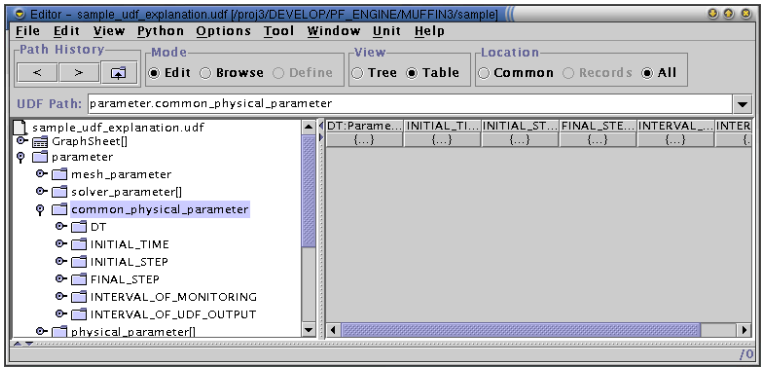


図 3.6: 共通物理パラメータ (UDF 解説)

- INITIAL_TIME: . . . タイムカウンターの初期値 (デフォルト値,0)
  ダイナミクスマネージャのタイムカウンターの初期値になる。
- FINAL_STEP: . . . 最大計算ステップ数。
  タイムステップカウンターの、このシミュレーションにおける最大値を指定。
  (注意: FINAL_STEP は 正の整数)
- INTERVAL_OF_MONITORING: . . . 計算状況モニタデータを出力するステップ幅
- INTERVAL_OF_UDF_OUTPUT: . . . 出力 UDF ファイルにレコードを出力するステップ幅

図 3.6 に、共通物理パラメータの GOURMET での表示を示す。

物理パラメータ (parameter.physical_parameter)

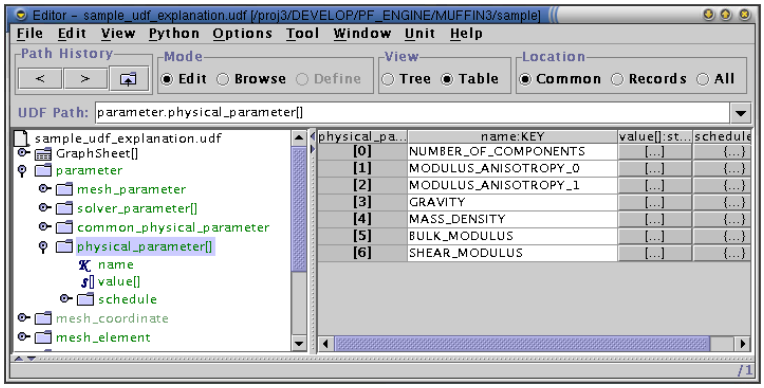


図 3.7: 物理パラメータ (UDF 解説)

ここに系を記述する物理化学的定数を入力する。物理パラメータに書かれる具体的なパラメータ名とその意味については、後の各々のシミュレータの説明部において述べるので、それらの章を参照にして欲しい。

図 3.7 に、Elastica での物理パラメータを例に示す。

パラメータの入力

以上で解説した

- ソルバ制御パラメータ (parameter.solver_parameter)
- 共通 物理パラメータ (parameter.common_physical_parameter)
- 物理パラメータ (parameter.physical_parameter)

のパラメータデータは同じ構造を持ち、入力手順も同じであるので、ここで解説する。

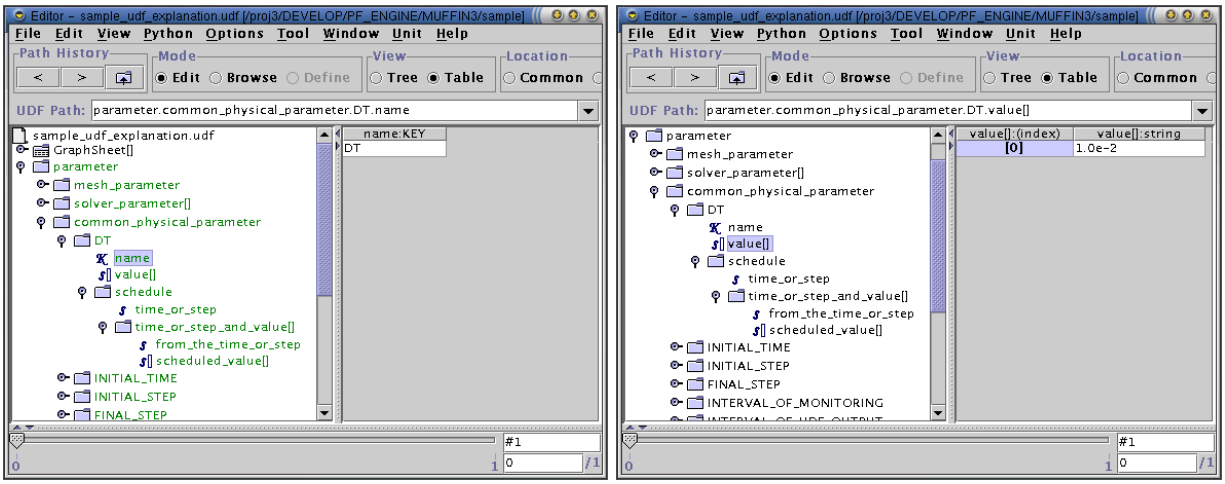


図 3.8: パラメータの入力 (UDF 解説)

パラメータデータの構造は以下の通り。

- name : . . . パラメータ名
  図 3.8 の左図は、パラメータ名 'DT' の場合。
- value[] : . . . パラメータの値 (文字列または数値) の配列
  図 3.8 の右図は、'DT' の値を $1.0e-2$ とする場合。
- schedule : . . . パラメータのスケジュールを記述

[規約 02]

パラメータ名に大文字・小文字の区別はない。
PARAMETER, parameter, Parameter, paraMeteR などは、すべて同じパラメータと扱われ重複は許されない。

[規約 03]

先頭以外の位置に任意の長さの空白を含む任意の長さのパラメータ名を使うことが可能である。
(たとえば、"`Seed of Random Number`") しかし、空白は間違いの元になるので空白としてアンダーバー記号 "_" を使うことを強く推奨する。(今回リリースエンジンも空白は使っていない筈) もちろん、全てが空白からなるパラメータは使えない. 空白から始まるパラメータはどこから始まるか曖昧なため使用不可とする。(ちなみに、入力部での前後の空白は、切り取っているので誤って空白がパラメータ名の前後に入っても問題無く認識される)

[規約 04]

Muffin では、1 つのパラメータに複数の値を配列で与えることが出来る。パラメータの値の型は、文字列・数値両方を使えるように文字列となってる。

[規約 05]

$(NameOfParameter) 変数について
あるパラメータが他のパラメータの値によって一意的に決まるパラメータで、かつ、入力 UDF に書かなくてはならない場合、$(NameOfParameter) でその値をパラメータの値に書けるようにした。例えば、成分の数を表す NUMBER_OF_COMPONENT というパラメータを 2 から 3 と変更した場合に dynamics_manager.registerd_field[].num_of_component(場の成分数) を 3 に書き変えなければならないのは面倒臭い。そこで $(NUMBER_OF_COMPONENT) と書くことでパラメータの値を参照できる (但し、パラメータ配列の先頭の値が参照される) ようにした。
$(NameOfParameter) の記述で、パラメータの値が参照できるのは、

- ソルバ制御パラメータ (parameter.solver_parameter)
- 共通 物理パラメータ (parameter.common_physical_parameter)
- 物理パラメータ (parameter.physical_parameter)
  の、すべてのパラメータの値を入れる領域 (スケジュール値の領域も含む)(但し、相互参照はエラーとなる。前方参照はもちろん後方参照もサポートしている。)
- 境界条件の値
  region_condition.condition[].value[]
- ダイナミクスマネージャの場の成分数の登録
  dynamics_manager.registerd_field[].num_of_component
- ダイナミクスマネージャのスケジュールの各種ステップやステップ幅
  dynamics_manager.schedule_of_simulation.evolution_plan.schedule[].from_this_time_step
  dynamics_manager.schedule_of_simulation.bifurcation_plan.interval_for_evaluation
  dynamics_manager.schedule_of_simulation.analysis_plan.interval_for_evaluation

[規約 06]

$(NameOfParameter) 変数の演算について
例えば、 $(NumberOfComponent)$(*)$(2) のように書けば自動的に計算展開するようにした。現在、$(*), $(/), $(+), $(-) が使用可能。
★演算順序に関する説明★
演算は前から 1 つずつ実行される。よって、A+B*C は → (A+B)*C と解釈される.

パラメータスケジューリングの入力

パラメータのスケジューリングとはユーザーが予め設定した複数の時刻 (or ステップ) でパラメータの値を自動的に変更する機能である。例えば、系の温度が時間的に変化するような系の相分離のシミュレーションを行いたい場合に、温度に依存するパラメータ (例えば、χ-パラメータ) を時間の関数として与えることが可能ならば、このような状況がすぐにシミュレーションにより再現出来る。例えば、共通物理パラメータ DT を 初期時刻で 1.0e-2 に設定し、タイムステップ 10000 で 5.0e-3 に、タイムステップ 20000 で 2.5e-3 に変更することも可能である。これは後期過程のダイナミクスの方が数値計算スキーム上不安定になりやすい場合に、DT を徐々に小さくして安定化をはかるといった場合に有効であろう。

以下にパラメータスケジュールデータの構造と、入力方法の参考として DT を STEP を使ってスケジュールした場合の Editor の図を示す。

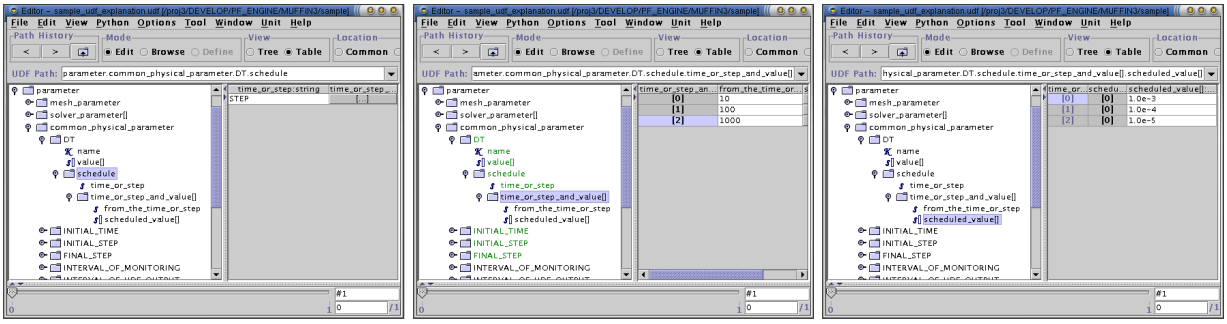


図 3.9: パラメータのスケジューリングの入力 (UDF 解説)

パラメータのスケジュールデータの構造は以下の通り。

- time_or_step：...TIME または STEP のいずれかを記述。
  スケジュールを時間でコントロールするか計算ステップでコントロールするかを指定。
  図 3.9 の左図は、パラメータ'DT' を STEP でスケジュールしたい場合。

- time_or_step_and_value[]：...スケジュールの配列

- time_or_step_and_value[].from_the_time_or_step：...パラメータ変更を行う時刻かステップを入力
  図 3.9 の中図は、パラメータ'DT' を 10,100,1000 STEP で 3 回変更したい場合。

- time_or_step_and_value[].scheduled_value[]：...パラメータの変更値
  図 3.9 の右図は、パラメータ'DT' を 10 STEP 目で $1.0e-3$ に、100 STEP 目で $1.0e-4$ に、1000 STEP 目で $1.0e-5$ に変更した場合。

3.1.3 メッシュ節点座標データ部 (mesh_coordinate)

すべての節点要素 (Vertex) の座標データが保持される。

メッシュ節点座標データは非構造格子の場合のみに出力される。メッシュパラメータのメッシュタイプを UNSTRUCTURED_INPUT とした場合には、ここのデータがエンジンに読み込まれメッシュが構築される。別のメッシングソフトをプリプロセッサとして利用し、メッシュを生成する場合には、このメッシュ節点座標データ部にプリポストで生成した節点座標データを入れておけば良い。

また、メッシュ節点座標データを、" コモンデータ" ・" レコードデータ" のどちらに出力するかもシミュレータのタイプによって異なる。流体系シミュレータ (Euler 描像) では" コモンデータ" に、固体系シミュレータ (Lagrange 描像) では" レコードデータ" より入出力される。[1]

以下に、データ構造を解説する。

- vertex[]：...系を構成するすべての節点要素の座標データ
  これらの要素の集合体は、場のデータ、ダイナミクスマネージャや python 解析プログラムなどで、
  "**ALL_VERTEX**" という名前で扱われる。
  - vertex[].id：...節点要素の ID(自然数：1,2,3,...)
  - vertex[].position：...節点要素の座標

[1] シミュレータのタイプに依るメッシュ節点座標データのコモン部およびレコード部への出力の振り分けは、Euler 描像では、コモンデータに出力され、Lagrange 描像、Adaptive Mesh では、レコードデータに出力される仕様である。これらは、コンパイルオプションでコントロールされる。詳細は、"B システム拡張方法" を参照。本リリースでは、具体的な Adaptive Mesh シミュレータは無いが、DynamicsManager,SuperMesh を始めとした MUFFIN の基本クラスライブラリではサポート済みである。

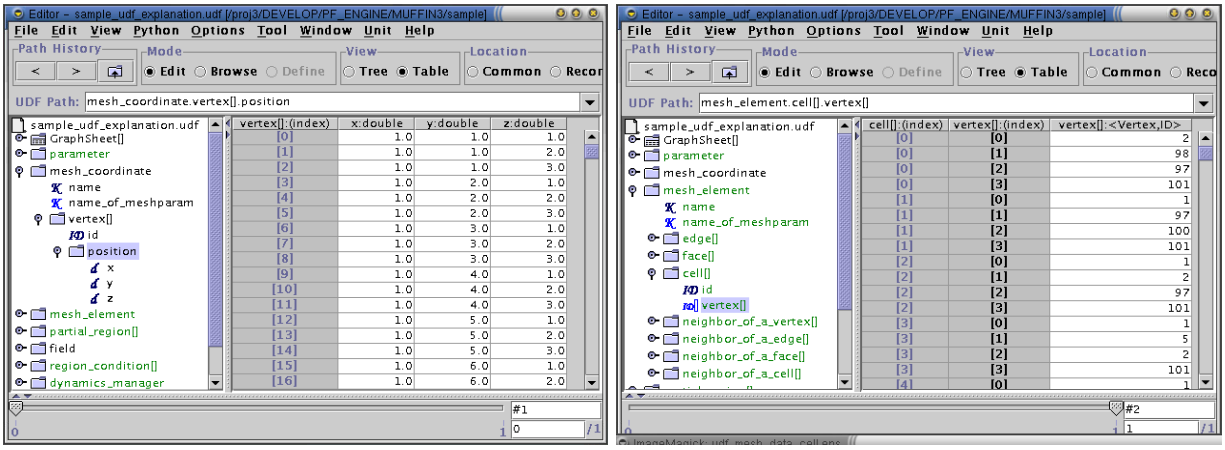


図 3.10: メッシュ構造データ (UDF 解説)

  - ∗ vertex[].position.x : . . . 節点の X 座標
  - ∗ vertex[].position.y : . . . 節点の Y 座標
  - ∗ vertex[].position.z : . . . 節点の Z 座標

  図 3.10 の左図参照。

- unit : . . . 座標の単位

3.1.4 メッシュ要素連結データ部 (mesh_element)

系全体を構成するメッシュ要素の連結データを保持する。すべての辺要素 (Edge) の節点連結情報、すべての面要素 (Face) の節点連結情報、すべてのセル要素 (Cell) の節点連結情報、近傍要素情報が保持される。

メッシュ要素連結データは非構造格子の場合のみに出力される。メッシュパラメータのメッシュタイプを UNSTRUCTURED_INPUT とした場合には、ここのデータがエンジンに読み込まれメッシュ要素が構築される。別のメッシングソフトをプリプロセッサとして利用し、メッシュを生成する場合には、このメッシュ要素連結データ部にプリポストで生成したメッシュ要素連結データを入れておけば良い。

常にすべてのデータが出力されるわけではなく、3 次元系の場合、系を構成する、すべてのセル要素 (Cell) の節点連結情報は必須であるが、辺要素 (Edge) の節点連結情報、面要素 (Face) の節点連結情報、近傍要素情報は、系を構成するすべての要素が出力されるわけではなく、各シミュレータでの必要性に任されている。同様に、2 次元系の場合、系を構成する、すべての面要素 (Face) の節点連結情報は必須であるが、辺要素 (Edge) の節点連結情報、近傍要素情報は、系を構成するすべての要素が出力されるわけではなく、各シミュレータでの必要性に任されている。[2]

また、メッシュ要素連結データは、本リリースシミュレータでは、すべて、” コモンデータ” より入出力される。[3]

以下に、データ構造を解説する。

- edge[] : . . . 系を構成するすべての辺要素の節点連結情報これらの要素の集合体は、場のデータ、ダイナミクスマネージャや python 解析プログラムなどで、”**ALL_EDGE**” という名前で扱われる。

[2] 今回リリースする 4 つのシミュレータは、いずれも近傍要素情報の入出力をしないので、常に近傍要素の UDF データは空になっている。近傍要素情報の入出力の有無はシミュレータのコンパイルオプションでコントロールされる。詳細は、”B システム拡張方法” を参照。

[3] シミュレータのタイプに依るメッシュ要素連結データのコモン部およびレコード部への出力の振り分けは、Euler 描像、Lagrange 描像では、コモンデータに出力され、Adaptive Mesh では、レコードデータに出力される仕様である。これらは、コンパイルオプションでコントロールされる。詳細は、”B システム拡張方法” を参照。本リリースでは、具体的な Adaptive Mesh シミュレータは無いが、MUFFIN の基本クラスライブラリではサポート済みである。

  - edge[].id : ... 辺要素の ID(自然数 : 1,2,3,...)
  - edge[].vertex : ... 辺要素の節点連結情報
    辺を両端を構成する 2 つの節点の ID が入る。

- face[] : ... 系を構成するすべての面要素の節点連結情報これらの要素の集合体は、場のデータ、ダイナミクスマネージャや python 解析プログラムなどで、"**ALL_FACE**" という名前で扱われる。
  - face[].id : ... 面要素の ID(自然数 : 1,2,3,...)
  - face[].vertex : ... 面要素の節点連結情報
    三角形要素の頂点を構成する 3 つの節点の ID が入る。

- cell[] : ... 系を構成するすべてのセル要素の節点連結情報これらの要素の集合体は、場のデータ、ダイナミクスマネージャや python 解析プログラムなどで、"**ALL_CELL**" という名前で扱われる。
  - cell[].id : ... セル要素の ID(自然数 : 1,2,3,...)
  - cell[].vertex : ... セル要素の節点連結情報
    四面体要素の頂点を構成する 4 つの節点の ID が入る。

  図 3.10 の右図参照。

3.1.5 部分領域データ部 (partial_region[])

Muffin では、ユーザーが自由にメッシュ要素の集合体を作成し集合体に名前をつけることで、部分領域や部分境界を構成することが出来る。

部分領域データは、常に" コモンデータ" から入出力される。メッシュパラメータのタイプが UNSTRUCTURED_INPUT の場合、メッシュ構造データと同様に部分領域データが読み込まれシミュレータ内部で部分領域が構築される。メッシュ構造データを入力せずに、シミュレータ内部でメッシュを生成する場合には、主な部分領域や部分境界の集合体がデフォルトで作成されるが、それ以外に独自の部分領域を追加したい場合には、この部分領域データに記述し、内部的に生成される部分領域とは異なる名前で登録すれば、部分領域データが読み込まれるしくみである。シミュレータに入力または内部で生成された部分領域データはすべて出力データの部分領域データに出力される。

部分領域 (要素の集合体) の名前のつけ方には、以下のルールを設けている。

1. "**ALL**" の文字列を含めば、系を構成するすべての要素、又は、すべての境界要素を表す。
2. "**BOUNDARY**" の文字列を含むか、先頭の文字が'**B**' または'**b**' ならは境界要素の集合体であることを表す。
3. "**VERTEX**" の文字列を含めば、節点要素の集合体を表す。
4. "**EDGE**" の文字列を含めば、辺要素の集合体を表す。
5. "**FACE**" の文字列を含めば、面要素の集合体を表す。
6. "**CELL**" の文字列を含めば、セル要素の集合体を表す。
   部分領域名には "**VERTEX**","**EDGE**","**FACE**", "**CELL**" の、どれか 1 つの文字列が含まれている必要がある。一方、これらのうち 2 つ以上の文字列を含むことは許されない。
7. 以上のルールに従い、次の部分領域名は予約語で再定義出来ない。
   (a) "**ALL_VERTEX**" ... すべての節点要素の集合体

(b) “**ALL_EDGE**” . . . すべての辺要素の集合体

(c) “**ALL_FACE**” . . . すべての面要素の集合体

(d) “**ALL_CELL**” . . . すべてのセル要素の集合体

(e) “**ALL_BOUNDARY_VERTEX**” . . . すべての境界節点要素の集合体

(f) “**ALL_BOUNDARY_EDGE**” . . . すべての境界辺要素の集合体

(g) “**ALL_BOUNDARY_FACE**” . . . すべての境界面要素の集合体

(h) “**ALL_BOUNDARY_CELL**” . . . すべての境界セル要素の集合体

以下に、データ構造を解説する。

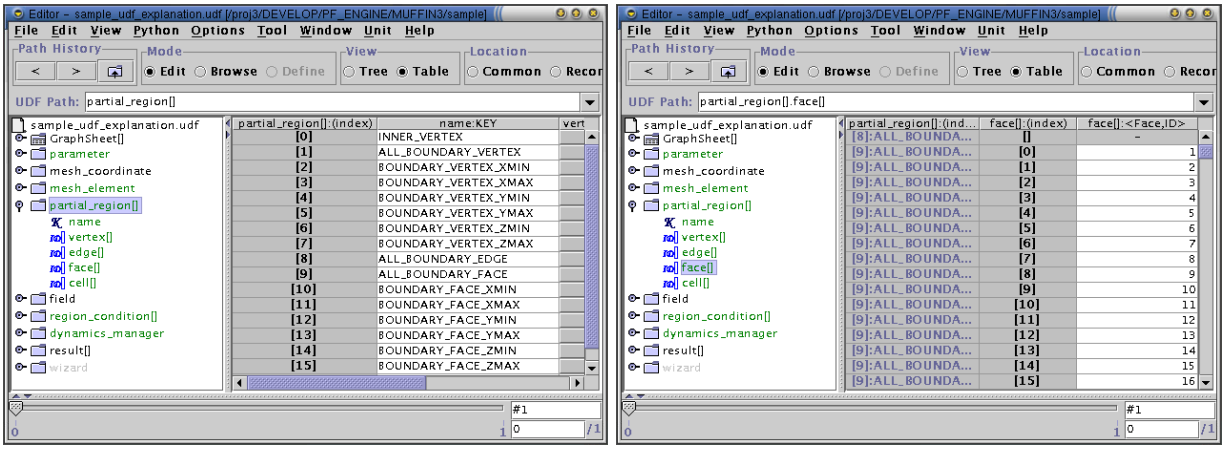


図 3.11: 部分領域データ - 部分領域名設定 - (UDF 解説)

- name : . . . 部分領域の名前を入力。
  命名のルールは上記の通り。図 3.11 の左図参照。
- vertex[] : . . . 部分領域を構成する節点要素の ID を入力
  部分領域名に”**VERTEX**” が含まれる場合に、ここのデータが有効となる。図 3.12 参照。
- edge[] : . . . 部分領域を構成する辺要素の ID を入力
  部分領域名に”**EDGE**” が含まれる場合に、ここのデータが有効となる。
- face[] : . . . 部分領域を構成する面要素の ID を入力
  部分領域名に”**FACE**” が含まれる場合に、ここのデータが有効となる。
- cell[] : . . . 部分領域を構成するセル要素の ID を入力
  部分領域名に”**CELL**” が含まれる場合に、ここのデータが有効となる。図 3.11 の右図参照。

3.1.6 場のデータ部 (field)

場のデータ部は、大きく分けて次の 3 つから構成される。

- スカラー場の配列 (field.scalar_field[])
- ベクトル場の配列 (field.vector_field[])

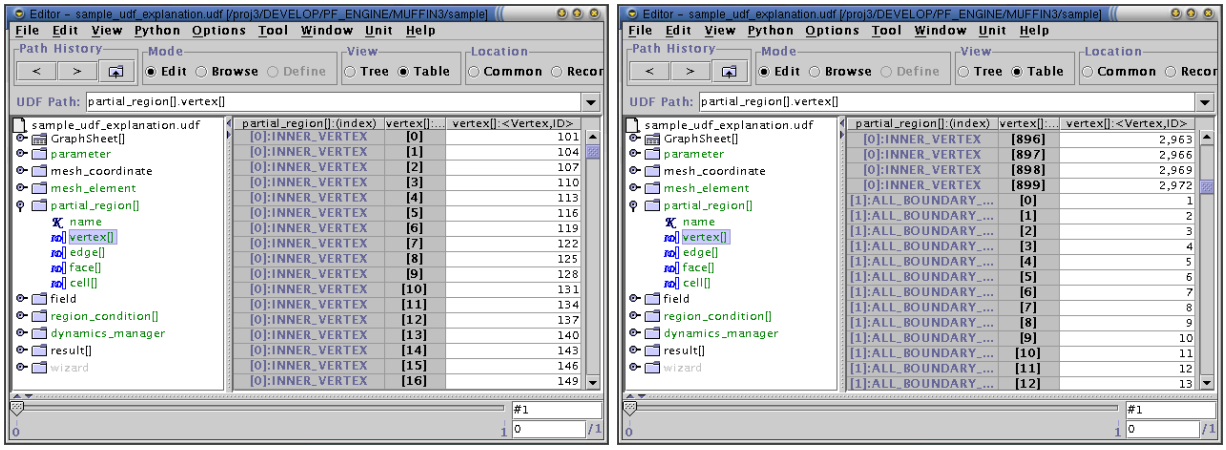


図 3.12: 部分領域データ - 部分領域要素設定 - (UDF 解説)

- テンソル場の配列 (field.tensor_field[])

場の値の入出力のルールは以下の通り、

- 入力
  通常のスタートの場合、コモンデータに該当する場のデータがあれば初期値として入力される。
  リスタートの場合、リスタートレコードに該当する場のデータがあれば初期値として入力される。

- 出力
  dynamics_manager.registered_field[].io_data_flag が 1 である場のデータが、すべてレコードデータに出力される。

以下に、各場のデータの構造を解説する。

スカラー場データ (field.scalar_field[])

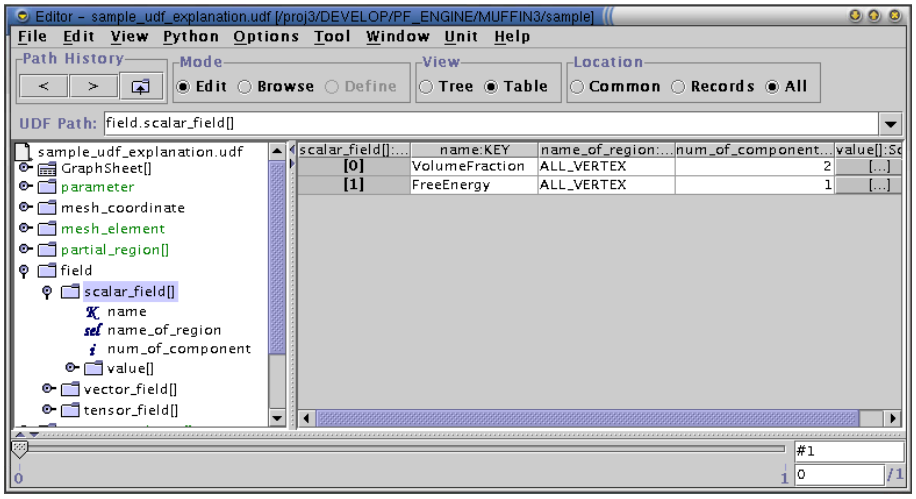


図 3.13: スカラー場データ - 場の名前・領域名 - (UDF 解説)

- name : . . . スカラー場の名前。 図 3.13 参照。

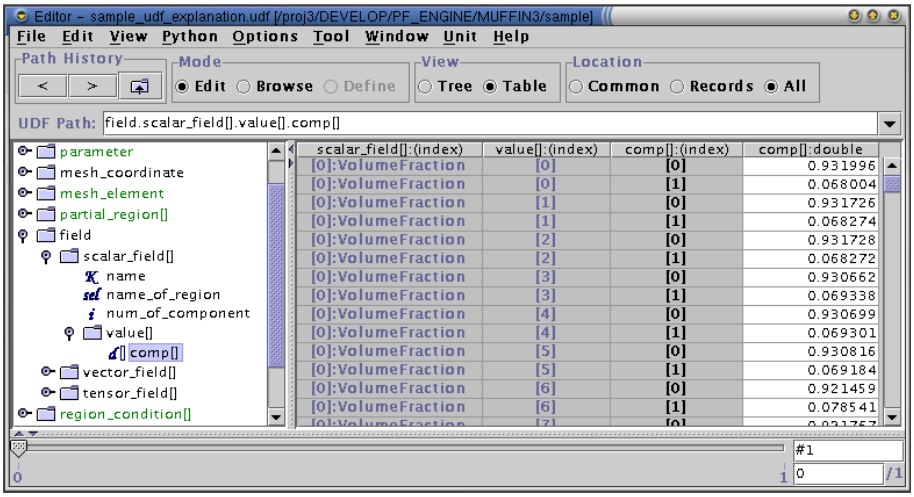


図 3.14: スカラー場データ - 場の値 - (UDF 解説)

- name_of_region : . . . この場が定義されている領域の名前
  上述の部分領域データにある部分領域名 (partial_region[].name)、又は、部分領域の予約語のいずれか。

- num_of_component : . . . 場の成分数
  場を定義する部分領域の要素 1 つ (1 節点、1 セルなど) についての場の値の数。

- value[]: . . . 場の値の配列 (メッシュ要素についての配列)
  場が定義されている部分領域データのメッシュ要素配列の順番で、各要素での場の値を保持する。

  - value[].comp[] : . . . 各メッシュ要素での場の値の配列 (成分についての配列)
    各要素でのスカラー場の値が成分数の長さの配列で保持される。図 3.14 参照。

ベクトル場データ (field.vector_field[])

- name : . . . ベクトル場の名前。 図 3.15 参照。

- name_of_region : . . . この場が定義されている領域の名前
  上述の部分領域データにある部分領域名、又は、部分領域の予約語のいずれか。

- num_of_component : . . . 場の成分数
  場を定義する部分領域の要素 1 つ (1 節点、1 セルなど) についての場の値の数。

- value[]: . . . 場の値の配列 (メッシュ要素についての配列)
  場が定義されている部分領域データのメッシュ要素配列の順番で、各要素での場の値を保持する。

  - value[].comp[] : . . . 各メッシュ要素での場の値の配列 (成分についての配列)
    各要素でのベクトル場の値が成分数の長さの配列で保持される。図 3.16 参照。

    * value[].comp[].x : . . . 各要素・各成分のベクトル場の X 成分の値
    * value[].comp[].y : . . . 各要素・各成分のベクトル場の Y 成分の値
    * value[].comp[].z : . . . 各要素・各成分のベクトル場の Z 成分の値

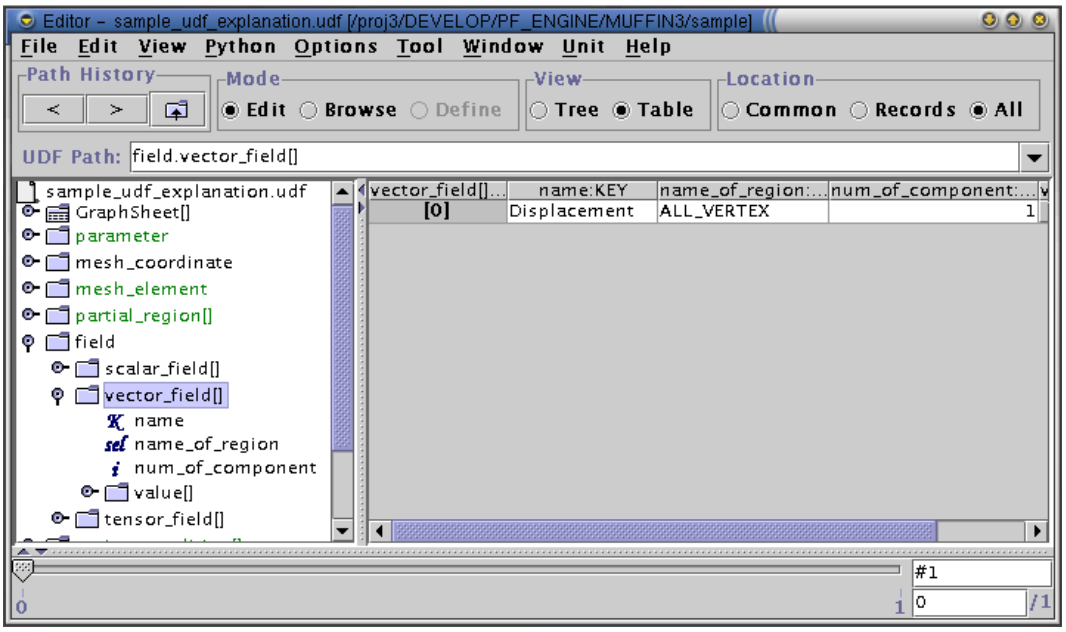


図 3.15: ベクトル場データ - 場の名前・領域名 - (UDF 解説)

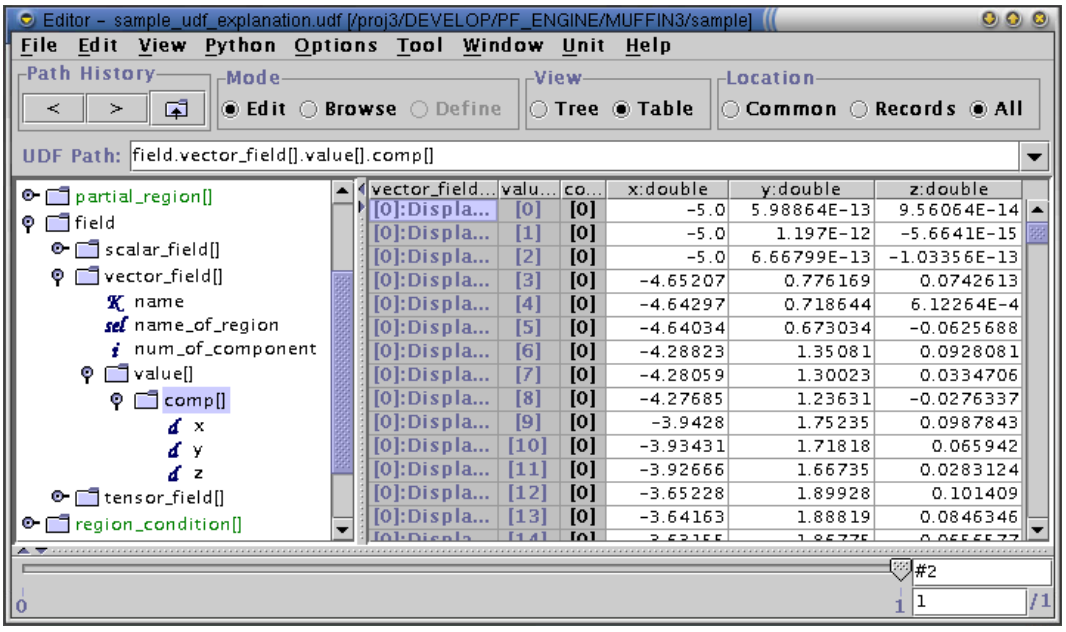


図 3.16: ベクトル場データ - 場の値 - (UDF 解説)

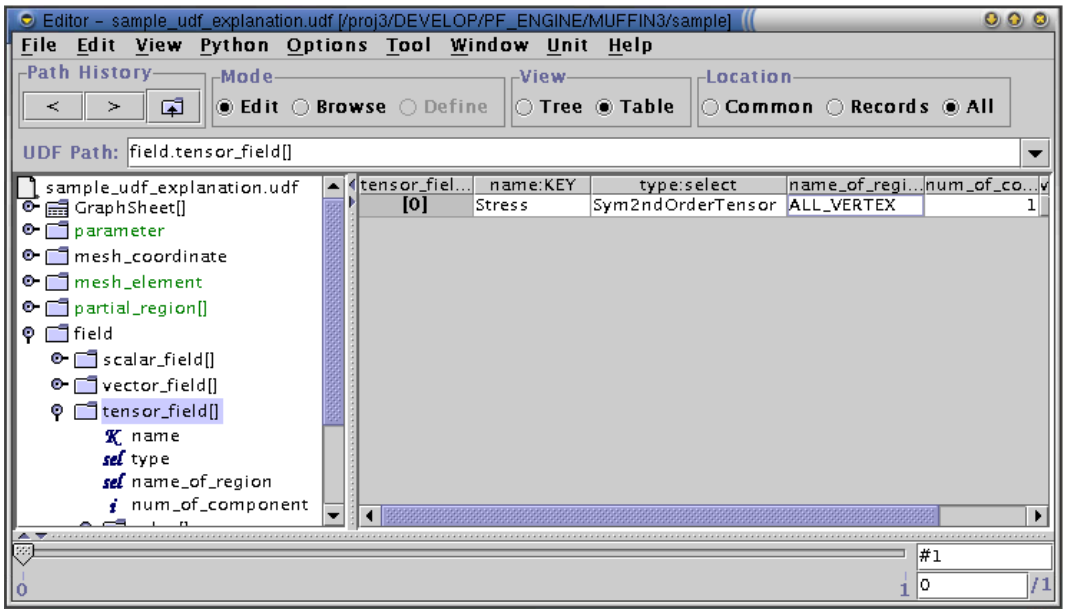


図 3.17: テンソル場データ - 場の名前・タイプ・領域名 - (UDF 解説)

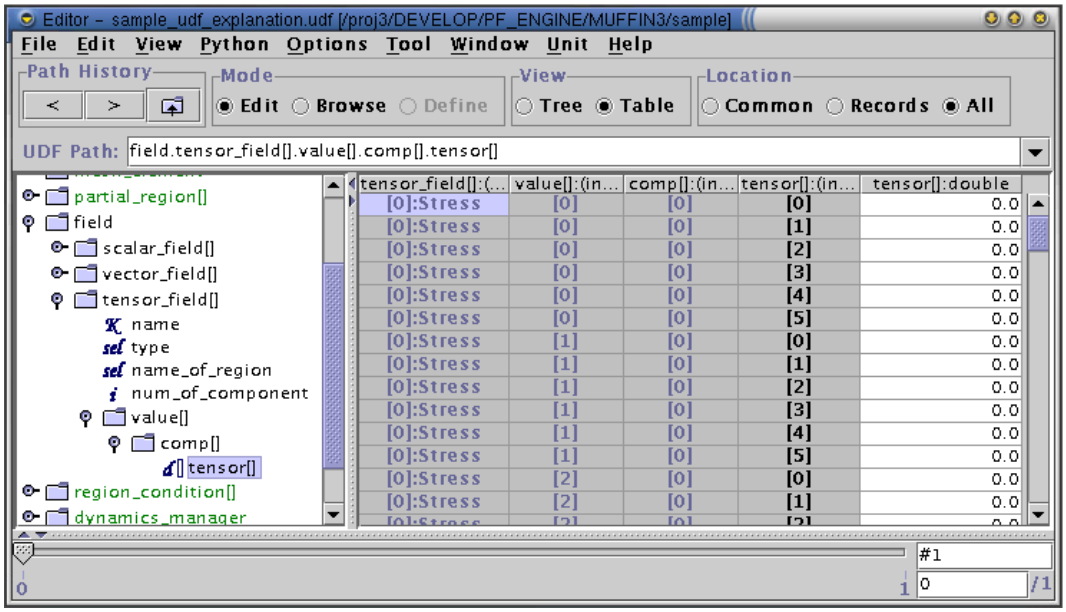


図 3.18: テンソル場データ - 場の値 - (UDF 解説)

テンソル場データ (**field.tensor_field[]**)

- name : . . . テンソル場の名前。図 3.17 は 2 階対称テンソル場の場合。
- type : . . . テンソルのタイプ。以下のいずれか。
  1. 2ndRankTensor . . . 2 階テンソル
  2. Sym2ndRankTensor . . . 2 階対称テンソル
  3. 3rdRankTensor . . . 3 階テンソル
  4. 4thRankTensor . . . 4 階テンソル
- name_of_region : . . . この場が定義されている領域の名前
  上述の部分領域データにある部分領域名、又は、部分領域の予約語のいずれか。
- num_of_component : . . . 場の成分数
  場を定義する部分領域の要素 1 つ (1 節点、1 セルなど) についての場の値の数。
- value[]: . . . 場の値の配列 (メッシュ要素についての配列)
  場が定義されている部分領域データのメッシュ要素配列の順番で、各要素での場の値を保持する。
  - value[].comp[] : . . . 各メッシュ要素での場の値の配列 (成分についての配列)
    各要素でのテンソル場の値が成分数の長さの配列で保持される。図 3.18 は 2 階対称テンソル場の場合。
    * value[].comp[].tensor[] : . . . 各要素・各成分のテンソル場の値を配列で保持。

3.1.7 部分領域 (境界) 条件データ部 (region_condition[])

MUFFIN では、部分領域 (境界) 条件データにより、任意の場の、任意の部分領域 (境界) について (境界) 条件をユーザーが設定出来る。このことと、部分領域 (境界) のユーザーによる自由な定義の機能が重なることで、ソルバの変更なく、さまざまな条件下でのシミュレーションが可能となっている。

部分領域条件は、主に以下の用途に用いられる。

- ある部分領域について、ある場の値を一定値で初期化する場合の初期化値の設定
- デリクレ、ノイマン、コーシーの境界条件の設定

以下に、データ構造と入力法を解説する。

- name : . . . 部分領域条件の識別名 (重複が無ければ何でも良い)
- name_of_region : . . . 条件を適用する部分領域 (境界) の名前。
  上述の部分領域データにある部分領域名 (partial_region[].name)、又は、部分領域の予約語のいずれか。
- name_of_field : . . . 条件を適用する場の名前
  シミュレータで有効な場 (dynamics_manager.registered_field[].name にある場) の名称のいずれかを入力。
- name_of_condition : . . . 部分領域 (境界) 条件の名前
  シミュレータでサポートされている (使用できる) 境界条件の名前は、各シミュレータの境界条件解説を参考にして下さい。境界条件の名前の命名には共通ルールとして、次の prefix を設けている。
  1. デリクレ (DIRICHLET) 境界条件. . . “**D_XXXXX**” と”**D_**” の prefix をつける。
  2. ノイマン (NEUMANN) 境界条件. . . “**N_XXXXX**” と”**N_**” の prefix をつける。

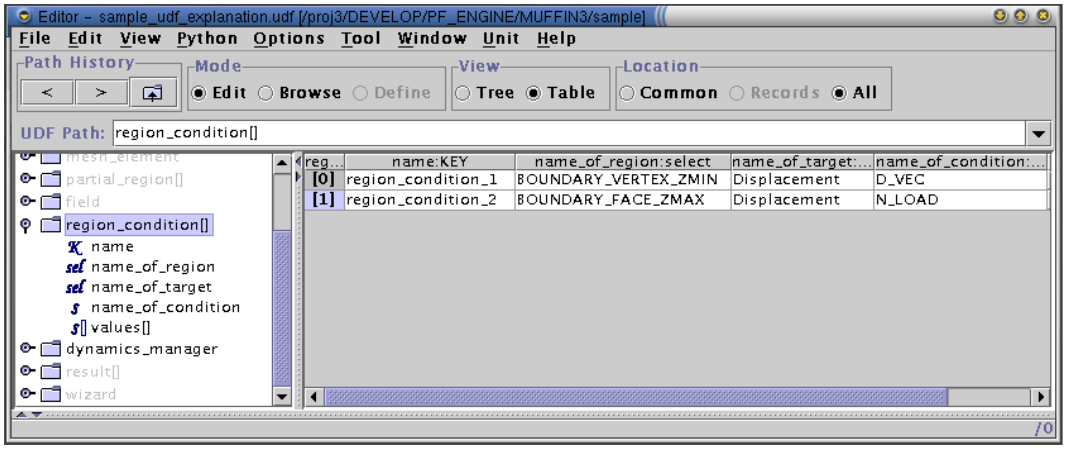


図 3.19: 部分領域 (境界) 条件データ - 領域名・場の名前・条件の名前 - (UDF 解説)

3. コーシー (CAUCHY) 境界条件... “**C_XXXXX**” と”**C_**” の prefix をつける。
4. 初期化条件... “**INIT_XXXXX**” と”**INIT_**” の prefix をつける。

以上について、Elastica で、Z 軸に垂直な底面”BOUNDARY_VERTEX_ZMIN” を固定 (変位場”Displacement” についてのデリクレ条件”D_VEC”) し、Z 軸に垂直な上面”BOUNDARY_FACE_ZMAX” に力を印可 (変位場”Displacement” についてのノイマン条件”N_LOAD”) する場合を図 3.19 に示す。

- values[] : ... 部分領域 (境界) 条件のパラメータ
  各条件に応じて必要なパラメータの数と意味は、各シミュレータの境界条件の解説を参考にしてください。

  1. デリクレ (DIRICHLET) 境界条件... デリクレ条件のパラメータ値を入力。1 自由度について 1 つ。
  2. ノイマン (NEUMANN) 境界条件... ノイマン条件のパラメータ値を入力。1 自由度について 1 つ。
  3. コーシー (CAUCHY) 境界条件... コーシー条件のパラメータ値を入力。1 自由度について 2 つ。
  4. 初期化条件... 場の初期化値を入力。1 自由度について 1 つ。

  以上について、Elastica で、Z 軸に垂直な底面”BOUNDARY_VERTEX_ZMIN” を固定 (変位場”Displacement” についてのデリクレ条件”D_VEC”、値は $(0.0, 0.0, 0.0)$) し、Z 軸に垂直な上面”BOUNDARY_FACE_ZMAX” に X 軸方向に 5.0 の力を印可 (変位場”Displacement” についてのノイマン条件”N_LOAD”、値は $(5.0, 0.0, 0.0)$) する場合を図 3.20 に示す。

3.1.8 解析結果データ部 (result[])

解析結果データには、各レコードで、シミュレータ内部で解析された結果のデータの名前と値が出力される。シミュレータに組み込まれている解析プログラムについては、3.2 ダイナミクスマネージャ解説の “動的解析実行” の手引、および、各シミュレータ部の解析コマンド一覧を参照。

解析結果データの構造は以下の通り。

- name : ... 結果データ名
- value[] : ... 結果データの値 (文字列または数値) の配列

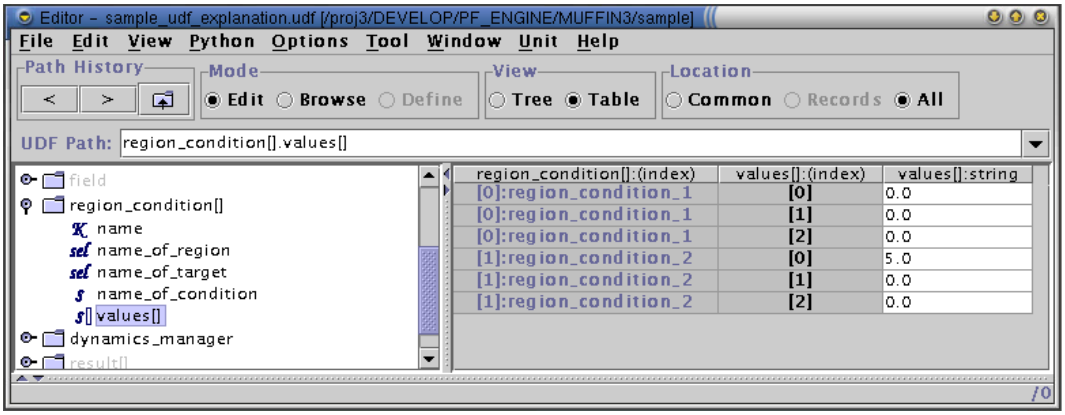


図 3.20: 部分領域 (境界) 条件データ - 条件の値 - (UDF 解説)

3.2 ダイナミクスマネージャ解説

ここでは、MUFFIN シミュレータの重要な基本機能であるダイナミクスマネージャの解説を行う。

MUFFIN でリリースする幾つかのシミュレータはすべて MUFFIN の基本機能であるダイナミクスマネージャ上に構築されている。また、UDF ファイルの入出力などはすべてダイナミクスマネージャに管理されているため、シミュレータによらず入出力 UDF 形式や使用法は共通になっている。

ダイナミクスマネージャの特徴は大きく分けて以下の 2 つである。

- スケジューリング機能
  パラメータやシミュレーションの時間スケージュール表を作成し、その計画に基づきシミュレーションを実行する。
- ダイナミクス構築機能
  連立して解く場や使う方程式、境界条件、初期状態などを選択し、適宜、ユーザーが解きたいシミュレータを組み上げることが出来る。

ダイナミクスマネージャデータの入出力 UDF ファイルでのデータ構造と入力法の解説の前に、ダイナミクスマネージャ用語定義とダイナミクスマネージャの機能解説を行う。

3.2.1 ダイナミクスマネージャ用語定義

ここでは、MUFFIN シミュレータの基本機能であるダイナミクスマネージャを理解する上で必要となる用語の解説を行う。

ダイナミクスマネージャの特徴は大きく分けて、上記の “スケジューリング機能”、” ダイナミクス構築機能” の 2 つである。このような機能は通常のシミュレータの利用形態 (パラメータ設定を行う) と基本的に異なっており、ダイナミクスマネージャでは、これらの機能を定義するために必要な用語を定義している。

ダイナミクスマネージャの 5 つの基本用語を以下に列挙する。

1. コマンド **(Command)**
2. プロシージャ **(Procedure)**
3. プロシージャテーブル **(ProcedureTable)**
4. 動的条件分岐 **(DynamicBifurcation)**

5. **動的解析実行 (DynamicAnalyzer)**
6. **スケジュール (Schedule)**
7. **ダイナミクスマネージング (DynamicsManaging)**

以下でこれらの用語の意味を詳しく解説する。

1. **コマンド (Command)** . . . 実行プログラムを構成する最小基本単位
   MUFFIN では、1 つのコマンドは、ある 1 つの場と場に送る 1 つのメッセージの組み合わせで構成されている。
   Command = an Operand (a_Field_object) + a Message
   ex.) Command = 化学ポテンシャル + Flory-Huggins 自由エネルギーを用いて計算
   このようにオペランド (ある場) とメッセージを合わせたものをコマンドと呼ぶ。MUFFIN ではコマンドは大きく、5 つのグループに分けられ登録されている。

   - 初期化用コマンド
   - 1 ステップ実行用コマンド
   - 境界条件設定用コマンド
   - 解析実行用コマンド
   - 評価実行用コマンド

   入力 UDF でプロシージャテーブル (後述) を作成する際にプロシージャ(後述) を作成するが、コマンドはこのプロシージャの 1 つの構成要素である。

2. **プロシージャ (Procedure)** . . . 処理手順の最小単位
   プロシージャとは、複数のコマンドを集めた集合体であり、処理手順の最小単位となる。1 つのプロシージャが実行されると、集合を形成する一連のコマンドが登録順に実行される。また、ある 1 つのプロシージャに対して名前を付けることが出来、そのプロシージャのことを **NamedProcedure** と呼ぶ。MUFFIN では **NamedProcedure** は大きく、次の 2 つのグループに分けられ登録される。

   - 初期化プロシージャ
   - (1 ステップ) 時間発展プロシージャ

   何れのグループに属するプロシージャも、評価実行用コマンドを除く 4 つのグループのコマンドのどれを用いて構築されても良い。

3. **プロシージャテーブル (ProcedureTable)** . . . プロシージャの集合体
   複数個の名前のつけられたプロシージャ**NamedProcedures** をある共通の意味の元にグルーピングしたもの。上述の通り、MUFFIN の **NamedProcedure** は、次の 2 つのプロシージャテーブルに分けられ登録される。

   - 初期化プロシージャテーブル
     初期化の方法にもいろいろあり、その一つ一つの手続きを **NamedProcedure** として定義し集めたもの。
   - (1 ステップ) 時間発展プロシージャテーブル
     様々な Dynamics の手続き (実行される順番の違いも含めて) を集めたもの。

4. **動的条件分岐 (DynamicBifurcation)** . . . ダイナミクスの動的変更
   シミュレーション中に一定時間間隔 (後述のフローチャートの BifurcationCheckTimeOrNot の所) で入力 UDF ファイルに予め入力した Bifurcation 評価コマンドを実行し、その返り値に応じて現在使用中の Procedure を別の Procedure に変更するしくみ。つまり、シミュレーションの条件分岐のしくみである。同時に、事前に時間スケジュールを入力している場合には、この時刻以降のスケジュールを破棄する事を指す。(Bifurcation 評価コマンドの返り値と、その値に応じて変更される Procedure は入力 UDF ファイルで ProceduresTable から選択しておく。)

   動的条件分岐を用いることで、平衡が必要となる系で、ある Bifurcation 評価コマンドにより平衡が達せられたかを評価し、その平衡に達した系に対し、何らかの操作 (例えば、電場や Shear 流動などを) を印加するなどが実現できる。或は、エネルギーがパラメータ E_o という値以下になった場合に別の解き方に変更するといった事も可能となる。

   MUFFIN 内部の実装としては、Bifurcation 評価コマンドは、エーテル場 (Ether Field) の評価実行用コマンドとして実装され提供されている。エーテル場は、入力 UDF で入力せずとも MUFFIN 内部でデフォルトで生成され保持されている場であり、入力されるすべての場を見ることが出来る。新しい Bifurcation 評価コマンドを登録したい場合には、エーテル場の評価実行用コマンドを追加すれば良い。(“B システム拡張方法” を参照。)

5. **動的解析実行 (DynamicAnalyzer)** . . . 解析プログラムの動的実行
   動的条件分岐と同様に、シミュレーション中に一定時間間隔 (後述のフローチャートの AnalysisCheckTimeOrNot の所) で入力 UDF ファイルに予め入力した複数の場の Analysis 評価コマンドを実行し、その返り値が真である場合のみ該当する場の対応する解析実行用コマンドを実行するしくみ。(Analysis 評価コマンドの選択やそれが真である場合に実行する解析実行用コマンドの選択は入力 UDF ファイルに記述しておく。)

6. **スケジュール (Schedule)** . . . シミュレーション実行計画 (タイムスケジュール)
   具体的には、

   - 初期化プロシージャを初期化プロシージャテーブルから 1 つ選択する。
   - 時間発展プロシージャテーブルから時間発展プロシージャ(a_Procedure) を 1 つ選び、その a_Procedure を何 STEP から何 STEP まで実行するか決める。そのような手続きを FINAL_STEP まで決定すること、
   - どのような 動的条件分岐、動的解析実行 を実行するかを決めること。

   これら 3 つを全て決定する事を指す。

7. **ダイナミクスマネージング (DynamicsManaging)** . . . Schedule の作成、実行、変更を行うこと。

3.2.2 ダイナミクスマネージャ機能解説

上述のダイナミクスマネージャ用語解説で、ダイナミクスマネージャの機能がやや見えたかと思う。ここでは、フローチャートを用いて、ダイナミクスマネージャの処理の流れと機能の使い方の概要を解説する。

MUFFIN シミュレータの処理の流れ

先ずは、入力 UDF ファイルを読み、パラメータ、メッシュ、場、ダイナミクスマネージャを構築して初期化、シミュレーションを実行し、出力データを生成、シミュレーション終了するまでの MUFFIN 共通の処理の流れを図 3.21 に、フローチャートで示す。

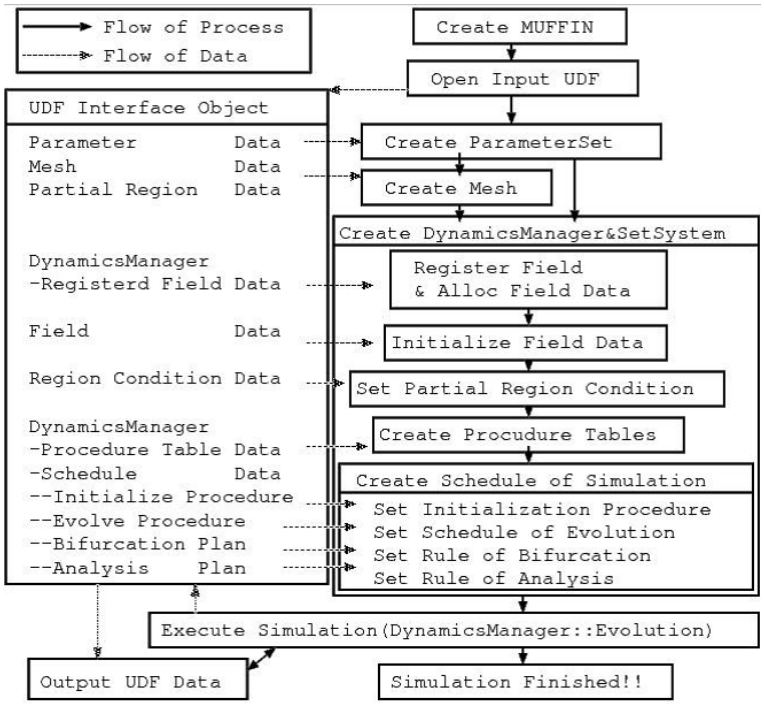


図 3.21: MUFFIN シミュレータの処理の流れ (ダイナミクスマネージャ解説)

ダイナミクスマネージャの処理の流れ

ダイナミクスマネージャを生成し、DynamicsManager::SetSystem 関数により初期化を行った後、DynamicsManager::Evolution 関数を実行すると、スケジュールに沿ったシミュレーションが自動的に実行される。

ダイナミクスマネージャの機能の理解のために、シミュレーション実行の中身の処理を、図 3.22 に、フローチャートで示す。

3.2.3 ダイナミクスマネージャ利用法概要

ダイナミクスマネージャのスケジューリング機能は、大きく分けて、次の 3 つのようなシミュレーションを行いたい場合に有効に働くことを想定して設計している。

3 通りのシミュレーション計画とは、

1. 定時刻プロシージャ変更型シミュレーション
2. 動的条件分岐型シミュレーション
3. 解析処理実行計画型シミュレーション

上の 3 通りのシミュレーション計画の作り方を簡単に説明する。

定時刻プロシージャ変更型シミュレーション計画

決まった時刻の間で実行するプロシージャを FINAL_STEP まで隙間無く決定することをいう。例えば、初期化法”A”(InitializeA) で初期化を行った後、シミュレーションの時刻 INITIAL_TIME からある時刻 TIME1 までは解き方”A”(EvolveA), 時刻 TIME1 からある時刻 TIME2 までは解き方”B”(EvolveB), 時刻 TIME2 から最終時刻 FINAL_TIME までは解き方”C”(EvolveC) でシミュレーションを実行する場合など。

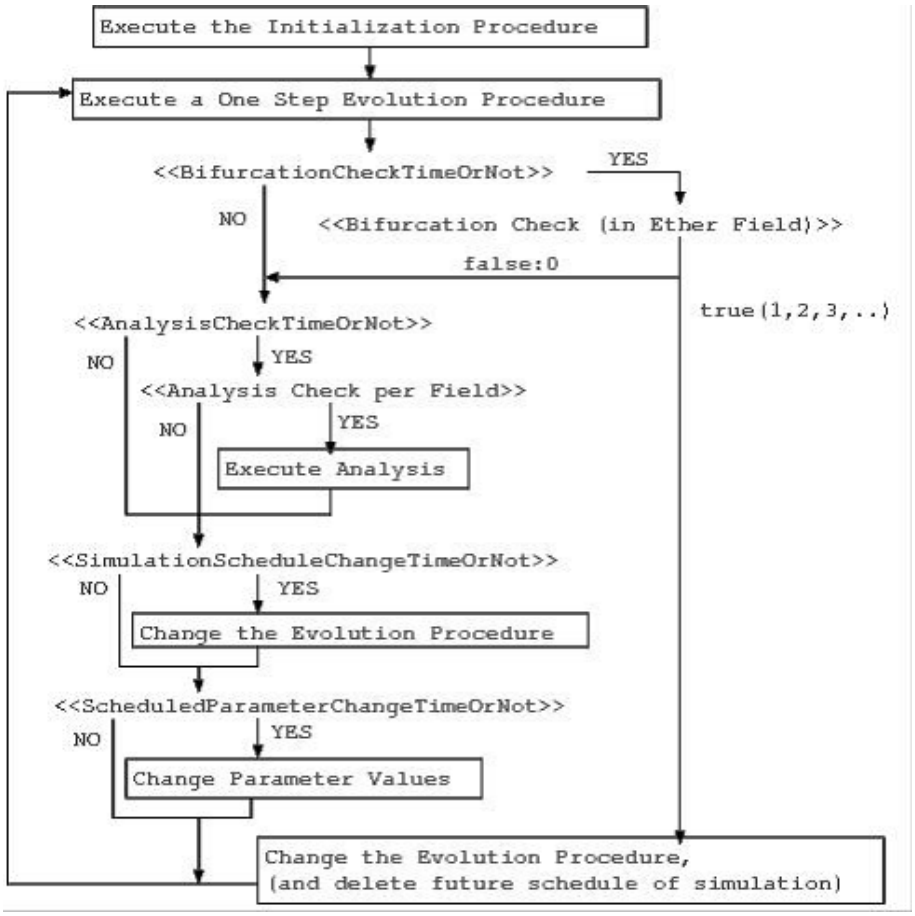


図 3.22: スケジュール実行処理の流れ (ダイナミクスマネージャ解説)

from_this_time_step	this_procedure_will_be_used
INITIALIZE	InitializeA
$(INITIAL_TIME)	EvolveA
TIME1	EvolveB
TIME2	EvolveC

表 3.1: 定時刻プロシージャ変更型シミュレーション計画 (ダイナミクスマネージャ解説)

選択する初期化プロシージャ(ここでは、InitializeA) は名前付きで初期化プロシージャテーブル (dynamics_manager.procedures_table_for_initialization[]) に、時間発展プロシージャ(ここでは、EvolveA,B,C) は名前付きで時間発展プロシージャテーブル (dynamics_manager.procedures_table_for_evolution[]) に、登録してあることが前提となる。

入力 UDF には dynamics_manager.evolution_plan.unit_of_schedule の欄に時間を使ってスケジュールを行うかステップを使ってスケジュールを行うかを記す. ステップ数を使ってスケジュールを行いたいならば、“**STEP**” と時間を使ってスケジュールを行いたいならば、“**TIME**” と書かなくてはならない。上記の例の場合、”TIME” を入力することになる。

初期化処理のプロシージャは、先頭データ dynamics_manager.evolution_plan.schedule[0].from_this_time_step のところに、”INITIALIZE” と書き込む。次に、dynamics_manager.evolution_plan.schedule[0].this_procedure_will_be_used に初期化プロシージャテーブル (dynamics_manager.procedures_table_for_initialization[]) から選択したプロシージャの名前を書き込む。

時間発展プロシージャは、2 番目のデータ dynamics_manager.evolution_plan.schedule[1].from_this_time_step のところに、プロシージャの実行を開始する時刻 0 (INITIAL_STEP や INITIAL_TIME を設定している場合には、その値、または、$(INITIAL_STEP) または $(INITIAL_TIME) でも良い) を書き込む。次に、dynamics_manager.evolution_plan.schedule[1].this_procedure_will_be_used に時間発展プロシージャテーブル (dynamics_manager.procedures_table_for_evolution[]) から選択したプロシージャの名前を書き込む。

もし、プロシージャを変更することの無いシミュレーションをしたいならこれで全て終了。これで FINAL_STEP までこのプロシージャが実行されることになる。後は自動的にこのスケジュールにしたがって実行されることになる。もし、途中の時刻でプロシージャを変更したいなら変更時刻を dynamics_manager.evolution_plan.schedule[i].from_this_time_step の欄に書き込み、その時刻以降に実行したいプロシージャを時間発展プロシージャテーブルから選び, そのプロシージャ名を dynamics_manager.evolution_plan.schedule[i].this_procedure_will_be_used の所に書き込む。このようにして、プロシージャ名とそのプロシージャに変更する時刻を次々と書き込めばスケージュールは完成する。あとは実行を行えばよい.

例えば、上の例の場合、表 3.1 のように書けばよい。

動的条件分岐型シミュレーション計画

例えば、初期化法”A”(InitializeA) で初期化を行った後、シミュレーションの時刻 INITIAL_TIME からある時刻 TIME1 までは解き方”A”(EvolveA), 時刻 TIME1 からある時刻 TIME2 までは解き方”B”(EvolveB), 時刻 TIME2 から最終時刻 FINAL_TIME までは解き方”C”(EvolveC) とシミュレーションを実行するという計画をしているのだが、一定時間間隔 TIME_D でユーザーが設定した評価関数”A”(EvaluateA) を評価し、それが返す評価値が 1(通常 0) ならばその時刻以降のシミュレーションの計画を破棄し、ある解き方”D”(EvolveD) に乗り換え実行、2(通常 0) ならば、その時刻以降のシミュレーションの計画を破棄し、ある解き方”E”(EvolveE) に乗り換え実行、3(通常 0) ならば、その時刻以降のシミュレーションの計画を破棄し、ある解き方”F”(EvolveF) に乗り換え実行、ということを実現したい場合に、動的条件分岐型シミュレーションが有効である。

return_value	this_procedure_...bifurcation
1	EvolveD
2	EvolveE
3	EvolveF

表 3.2: 動的条件分岐型シミュレーション計画 (ダイナミクスマネージャ解説)

選択する初期化プロシージャ(ここでは、InitializeA) は名前付きで初期化プロシージャテーブル (dynamics_manager.procedures_table_for_initialization[]) に、時間発展プロシージャ(ここでは、EvolveA,B,C,D,E,F) は名前付きで時間発展プロシージャテーブル (dynamics_manager.procedures_table_for_evolution[]) に、存在しなければならない。また、条件分岐評価関数 (ここでは、EvaluateA) がエーテル場 (Ether Field) の評価関数として登録してあることが前提となる。

これらのスケジュールを設定するには、先ず、上述の定時刻プロシージャ変更型シミュレーション計画を設定する。更に、入力 UDF には dynamics_manager.bifurcation_plan.unit_of_schedule の欄に時間を使ってスケジュールを行うかステップを使ってスケジュールを行うかを記す．ステップ数を使ってスケジュールを行いたいならば、“**STEP**” と時間を使ってスケジュールを行いたいならば、“**TIME**” と書かなくてはならない。上記の例の場合、”TIME” を入力する。続いて、条件分岐評価関数の評価時間間隔 TIME_D を、dynamics_manager.bifurcation_plan.interval_for_evaluation に入力し、条件分岐評価関数の名前 EvaluateA を、dynamics_manager.bifurcation_plan.evaluation_function_name に入力する。

続いて、評価関数の返り値と、分岐する時間発展プロシージャの関連を記述する。
bifurcation_plan.bifurcation_candidate[i].return_value に先頭から返り値の $1, 2, 3, ...$ を入力し、
bifurcation_plan.bifurcation_candidate[i]. this_procedure_will_be_used_for_evolution_after_bifurcation に返り値の値に対応して、分岐する時間発展プロシージャの名前を記述する。

例えば、上の例の場合、表 3.2 のように書けばよい。

解析処理実行計画型シミュレーション計画

ある場に対して実行したい解析関数の実行計画を記述できる。計画には、以下の情報を記述する。

- 解析評価関数を実行する時間間隔の単位 (unit_of_schedule)
- 解析評価関数を実行する時間間隔 (interval_for_evaluation)
- 場の名前 (analyzer_list[].field)
- 解析評価関数の名前 (analyzer_list[].evaluation_function_name)
- 解析用関数の名前 (analyzer_list[].analysis_function_name)

これらの情報を用いて次のようなシミュレーション計画を構築出来る。

1. 定時刻解析実行型
   評価関数として常に真 (非ゼロ) を返す関数を用いることにより実現する。
2. 条件判断による解析実行計画
   例えば、1000 ステップ毎に、場”nameOfField_X” の 2 つの評価関数”nameOfEvalFunc_A” と ”nameOfEvalFunc_B” を、場”nameOfField_Y” の 2 つの評価関数”nameOfEvalFunc_C” と ”nameOfEvalFunc_D” を、場”nameOfField_Z” の評価関数”nameOfEvalFunc_E” を実行し、各々の評価値が真であれば、対応する場の解析関数を実行したい場合、次のように記述する。

field	evaluation_function_name	analysis_function_name
nameOfField_X	nameOfEvalFunc_A	nameOfAnalyzerFunc00
nameOfField_X	nameOfEvalFunc_B	nameOfAnalyzerFunc01
nameOfField_Y	nameOfEvalFunc_C	nameOfAnalyzerFunc11
nameOfField_Y	nameOfEvalFunc_D	nameOfAnalyzerFunc12
nameOfField_Z	nameOfEvalFunc_E	nameOfAnalyzerFunc23

表 3.3: 動的解析実行型シミュレーション計画 (ダイナミクスマネージャ解説)

- analysis_plan.unit_of_schedule に"STEP" を記述。
- analysis_plan.interval_for_evaluation に 1000 を記述。
- analysis_plan.analyzer_list[] の field, evaluation_function_name, analysis_function_name には、場の名前、解析評価関数の名前、評価値が真の場合に実行する解析関数の名前を表 3.3 のように入力する。

3.2.4 入出力 UDF：ダイナミクスマネージャ部 (dynamics_manager) 詳細解説

ダイナミクスマネージャ部 (dynamics_manager) の概要

入力 UDF ファイルのダイナミクスマネージャ部で、シミュレーションしたい系のダイナミクスを組み立て、シミュレーションスケジュールを記述する。

出力 UDF ファイルのダイナミクスマネージャ部には、入力のダイナミクスマネージャデータがそのまま出力され、結果データと一緒にシミュレーションで使用したダイナミクスやスケジュールが記録される。リスタートの際に出力 UDF ファイルのダイナミクスマネージャ部が読み込まれるので、ダイナミクスやスケジュールを変更したい場合にはリスタート前に編集の必要がある。

入出力 UDF ダイナミクスマネージャ部は、大きく分けて以下の 4 つからなる。

1. 登録する場 (dynamics_manager.registered_field[])
   マニュアルに書かれた使用可能な場のリストの中から、シミュレーションで使う場を登録する。

2. 初期化プロシージャテーブル (dynamics_manager.procedures_table_for_initialization[])
   登録した場、各々に対して実行する初期化手続きの候補 (初期化プロシージャ) を取りまとめ、登録を行うところ。

3. 時間発展プロシージャテーブル (dynamics_manager.procedures_table_for_evolution[])
   (1 ステップ) 時間発展の際に" 登録した場" に対して実行するコマンドの一群を順序付で作成し、それを時間発展プロシージャとして名前を付けて登録する。複数の時間発展プロシージャを登録することが可能ある。複数個のプロシージャを使って、後で説明を行うシミュレーションのスケージュール作成することが可能である。

4. スケジュール (dynamics_manager.schedule_of_simulation)
   シミュレーションのスケージュールを記述する。次の 4 つの項目を必要に応じて入力する。

   (a) 初期化プロシージャの選択
   初期化プロシージャテーブルの中から、使いたい初期化を選択する。

   (b) 時間発展プロシージャの選択とタイムスケジュール
   時間発展プロシージャテーブルの中から使いたい時間発展プロシージャを選択し、ステップか時間を使ってどの時刻からどの時刻までこのプロシージャを使うか決定する。これを最終時刻 (FINAL_STEP or FINAL_TIME) まで決定する。

(c) 動的条件分岐
シミュレーションの分岐を実行するために以下のような情報を記述する.

i. 条件分岐のための用いる評価コマンドの名前。
ii. 評価コマンドの返り値に応じて時間発展プロシージャを変更するための、返り値と時間発展プロシージャの対応関係。
iii. 評価コマンドを実行する時間 (または STEP) 間隔。

(d) 動的解析実行
解析したい場と解析したい方法 (解析実行用コマンド) を選択し, 解析を実行するかどうかを決める評価コマンドと共に記述する。複数の解析を実行可能。

登録する場 (dynamics_manager.registered_field[])

ダイナミクスマネージャの登録する場の記述法を解説する。ここには、シミュレーションで必要な複数の場を登録する。各々の登録する場について、場の名前やタイプ、成分数、出力データの有無などを登録する。
以下に、データ構造と入力法を解説する。

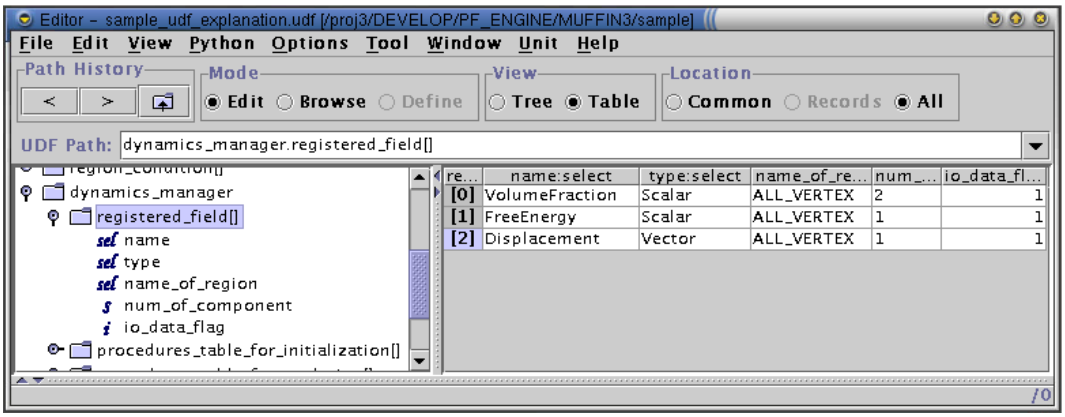


図 3.23: ダイナミクスマネージャ - 登録する場 - (UDF 解説)

- name : . . . 場の名前
  場の名前は、各シミュレータのリファレンスの” 利用可能な場の一覧” より選択。
- type : . . . 場の変数の型。以下のいずれか 1 つを選択する。
  1. Scalar . . . スカラー場
  2. Vector . . . ベクトル場
  3. 2ndRankTensor . . . 2 階テンソル
  4. Sym2ndRankTensor . . . 2 階対称テンソル
  5. 3rdRankTensor . . . 3 階テンソル
  6. 4thRankTensor . . . 4 階テンソル
- name_of_region : . . . 場が定義されている領域の名前
  部分領域データにある部分領域名 (partial_region[].name)、又は、以下の部分領域の予約語のいずれか。
  1. “**ALL_VERTEX**” . . . すべての節点要素の集合体

場の名前	場の型	場の領域	成分数	出力データの有無
VolumeFraction	Scalar	ALL_VERTEX	2	有
FreeEnergy	Scalar	ALL_VERTEX	1	有
Displacement	Vector	ALL_VERTEX	1	有

表 3.4: ダイナミクスマネージャ - 登録する場 - (UDF 解説)

2. “**ALL_EDGE**” . . . すべての辺要素の集合体
3. “**ALL_FACE**” . . . すべての面要素の集合体
4. “**ALL_CELL**” . . . すべてのセル要素の集合体
5. “**ALL_BOUNDARY_VERTEX**” . . . すべての境界節点要素の集合体
6. “**ALL_BOUNDARY_EDGE**” . . . すべての境界辺要素の集合体
7. “**ALL_BOUNDARY_FACE**” . . . すべての境界面要素の集合体
8. “**ALL_BOUNDARY_CELL**” . . . すべての境界セル要素の集合体

- num_of_component : . . . 場の成分数
  場を定義する部分領域の要素 1 つ (1 節点、1 セルなど) についての場の値の数。通常、多成分で確保したい場合、成分数は パラメータ NUMBER_OF_COMPONENT の値と一致することが多いであろう。その場合には、“$NUMBER_OF_COMPONENT” と入力しておけば、パラメータの値が自動的に読み込まれる。

- io_data_flag : . . . 出力 UDF ファイルへの、この場のデータの出力の有無。
  ”1” 必要 、”0” 不必要で、”1” の場合には、出力 UDF ファイルに場のデータが出力される。リスタートの際に再入力したい場のデータは、出力しておくこと。

以上の 6 つの情報を各々の登録する場に対して与える。入力例を、図 3.23 に示す。
図 3.23 は、表 3.4 の 3 つの場を登録する例である。
但し、通常ユーザーは、各シミュレータの入力テンプレート UDF ファイルやサンプル UDF ファイルを編集し、入力 UDF ファイルを作成することになるため、これらのデータを考えていれる必要は無い。

初期化と時間発展のプロシージャーテーブル

ダイナミクスマネージャの初期化および時間発展プロシージャテーブルの記述法を解説する。ここでは、1 つ以上の初期化プロシージャ, 時間発展プロシージャをそれぞれ記述する。これらのテーブルを用い、以下で解説するシミュレーションスケジュールを構築することが可能となる。よってこのテーブル作成は、シミュレーションの (1 ステップ) 時間発展の候補を作成するということになる。
初期化プロシージャテーブル (dynamics_manager.procedures_table_for_initialization[]) と、時間発展プロシージャテーブル (dynamics_manager.procedures_table_for_evolution[]) のプロシージャテーブルの構造は同じであるので、データ構造と入力法を合わせて解説する。

プロシージャテーブルはプロシージャの配列である。1 つのプロシージャは、プロシージャ名と一連のコマンド列 (場の名前とメッセージの名前) を入力することで作成する。
ここでは、初期化プロシージャとして、表 3.5 の 4 つのプロシージャを初期化プロシージャテーブルに登録する場合を図 3.24 に示し解説する。
また、時間発展プロシージャとして、表 3.6 の 2 つのプロシージャを時間発展プロシージャテーブルに登録する場合を図 3.25 に示し解説する。
以下に、プロシージャのデータ構造と入力法を解説する。

[1]	初期化プロシージャ名	INITIALIZE:LINEAR_ELASTICITY:ISOTROPIC:UNIFORM
実行順	場の名前	実行する処理 (メッセージ) の名前
1	VolumeFraction	INITIALIZE:UNIFORM
2	Displacement	INITIALIZE:TO_ZERO
3	FreeEnergy	SOLVE:LINEAR_ELASTICITY
[2]	初期化プロシージャ名	INITIALIZE:LINEAR_ELASTICITY:ISOTROPIC:ONE_COMPONENT
実行順	場の名前	実行する処理 (メッセージ) の名前
1	VolumeFraction	INITIALIZE:ONE_COMPONENT
2	Displacement	INITIALIZE:TO_ZERO
3	FreeEnergy	SOLVE:LINEAR_ELASTICITY
[3]	初期化プロシージャ名	INITIALIZE:LINEAR_ELASTICITY:ISOTROPIC:TWO_COMPONENT
実行順	場の名前	実行する処理 (メッセージ) の名前
1	VolumeFraction	INITIALIZE:TWO_COMPONENT
2	Displacement	INITIALIZE:TO_ZERO
3	FreeEnergy	SOLVE:LINEAR_ELASTICITY
[4]	初期化プロシージャ名	INITIALIZE:LINEAR_ELASTICITY:ISOTROPIC:MULTI_COMPONENT
実行順	場の名前	実行する処理 (メッセージ) の名前
1	Displacement	INITIALIZE:TO_ZERO
2	FreeEnergy	SOLVE:LINEAR_ELASTICITY

表 3.5: ダイナミクスマネージャ - 初期化プロシージャ例 - (UDF 解説)

[1]	時間発展プロシージャ名	SOLVE:LINEAR_ELASTICITY:ISOTROPIC
実行順	場の名前	実行する処理 (メッセージ) の名前
1	Displacement	SOLVE:LINEAR_ELASTICITY:ISOTROPIC
2	FreeEnergy	SOLVE:LINEAR_ELASTICITY
3	Displacement	MOVE:POSITION_OF_VERTEX
[2]	時間発展プロシージャ名	SOLVE:LINEAR_ELASTICITY:ANISOTROPIC
実行順	場の名前	実行する処理 (メッセージ) の名前
1	Displacement	SOLVE:LINEAR_ELASTICITY:ANISOTROPIC
2	FreeEnergy	SOLVE:LINEAR_ELASTICIT
3	Displacement	MOVE:POSITION_OF_VERTEX

表 3.6: ダイナミクスマネージャ - 時間発展プロシージャ例 - (UDF 解説)

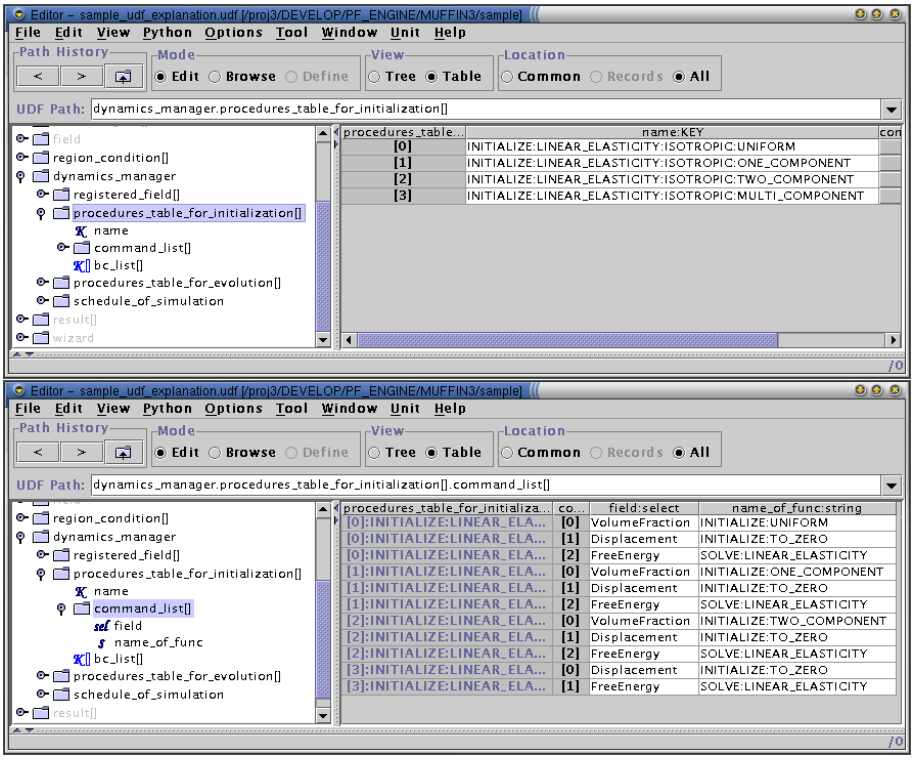


図 3.24: ダイナミクスマネージャ - 初期化プロシージャテーブル - (UDF 解説)

- name：. . . プロシージャの名前
  処理の内容が分かりやすい名前を推奨する。図 3.24 の上図に、初期化プロシージャ名の入力例を図 3.25 の上図に、時間発展プロシージャ名の入力例を示す。

- command_list[]: . . . このプロシャージャで実行する一連のコマンド
  登録順に実行されるので、入力の順番に注意。1 つのコマンドは、以下のデータで構成される。

  – command_list[].field：. . . 場の名前
    上述の” 登録する場” で登録した場の名前を記述

  – command_list[].name_of_func：. . . 実行する処理 (メッセージ) の名前を記述
    マニュアルに書かれた各々の場のコマンド一覧の中から、コマンド名を選択し登録する。

  図 3.24 の下図に、初期化プロシージャのコマンドの入力例を図 3.25 の下図に、時間発展プロシージャのコマンドの入力例を示す。

スケジュール (dynamics_manager.schedule_of_simulation) の記述

ここでは、ダイナミクスマネージャのシミュレーションスケジュールの記述法について解説する。

ここは、シミュレーションのスケジュールについての全ての情報を記述するところであり、シミュレーションで用いる、初期化プロシージャと時間発展プロシージャの選択、必要なら時間発展プロシージャのタイムスケジュール、動的条件分岐、動的解析実行を入力する。

以下に、スケジュールの入力データ構造の概要と記述法を解説する。

- name：. . . スケジュールの名前 (何でも良い)
  入力例を図 3.26 に示す。

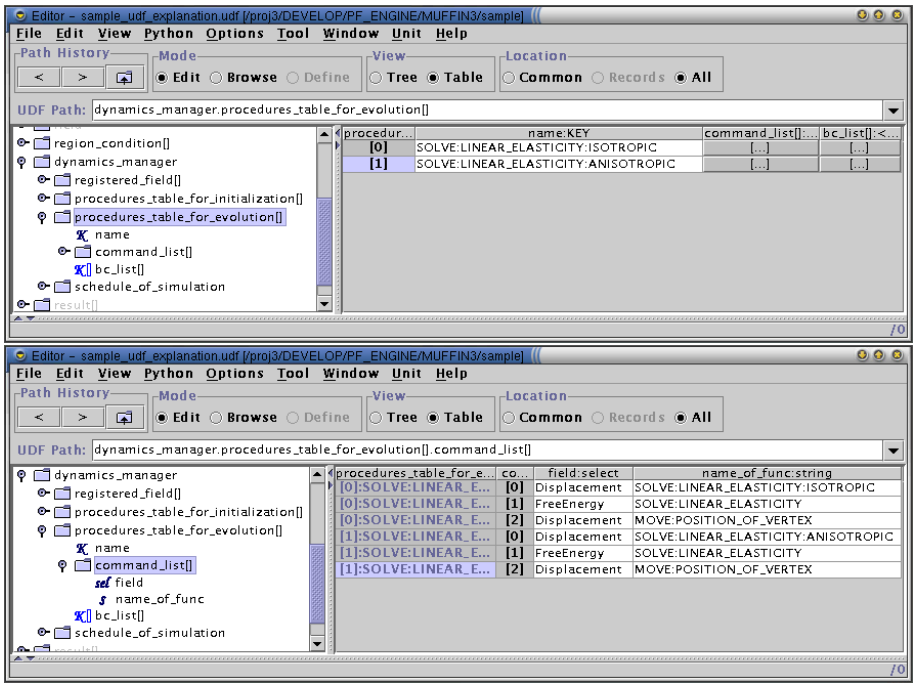


図 3.25: ダイナミクスマネージャ - 時間発展プロシージャテーブル - (UDF 解説)

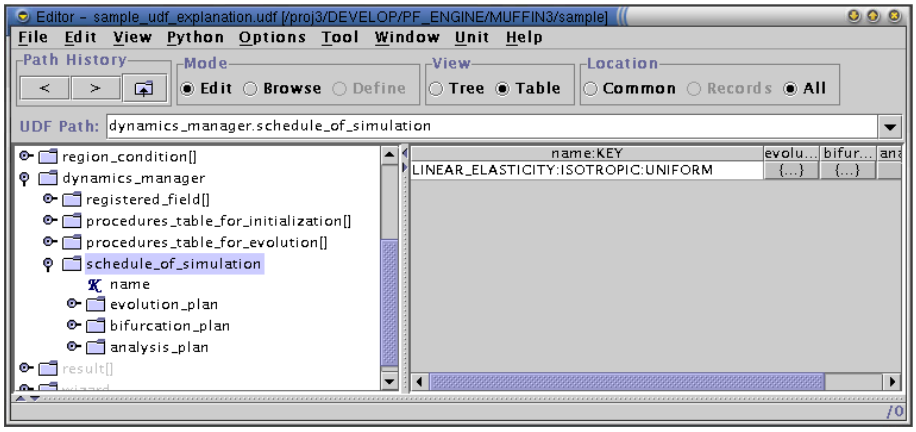


図 3.26: ダイナミクスマネージャ - スケジュールの名前 - (UDF 解説)

- evolution_plan：...初期化プロシージャの選択と定時刻プロシージャ変更型シミュレーションの記述
  - evolution_plan.unit_of_schedule：...タイムスケジュールに用いる単位
    “TIME” または”STEP” のいずれかを入力する。
    1. “TIME” ...時刻でスケジューリングを行う場合
    2. “STEP” ...計算ステップでスケジューリングを行う場合

    計算ステップでスケジュールを行う場合の入力例を、図 3.27 に示す。

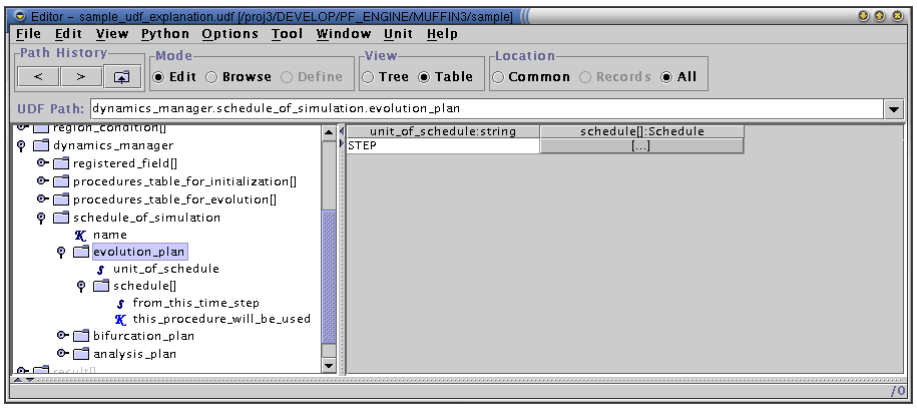


図 3.27: ダイナミクスマネージャ - タイムスケジュールの単位 - (UDF 解説)

  - evolution_plan.schedule[]：...初期化プロシージャと時間発展プロシージャの選択。
    配列の先頭データで初期化プロシージャの選択を行い、2 番目以降のデータに、時間発展プロシージャの実行スケジュールを記入する。スケジュールデータは、開始時刻または開始 STEP と、実行するプロシージャの名前の組み合わせで成り立っており、これらのデータの配列として、スケジュールを登録する。初期化プロシージャについては、開始時刻または開始 STEP データに”INITIALIZE” と文字列で入力する。
    - evolution_plan.schedule[].from_this_time_step：...開始時刻または開始 STEP
      スケジュール実行を行うプロシージャについて、evolution_plan.unit_of_schedule が “TIME” の場合、double 型の開始時刻を、”STEP” の場合、int 型の開始 STEP を入力する。初期化プロシージャの選択を行う先頭データについては、”INITIALIZE” という文字列を入力する。
    - evolution_plan.schedule[].this_procedure_will_be_used：...実行プロシージャ名
      実行するプロシージャ名を入力する。先頭データの初期化プロシージャの選択では、初期化プロシージャテーブルに登録されているデータから 1 つのプロシージャを選択し名前を入力する。2 番目以降の時間発展スケジュールについては、時間発展プロシージャテーブルに登録されているデータからプロシージャを選択し名前を入力する。

    入力例を図 3.28 に示す。入力例は初期化プロシージャとして、
    ”INITIALIZE:LINEAR_ELASTICITY:ISOTROPIC:TWO_COMPONENT” を、1STEP から最終ステップまでの時間発展プロシージャにすべて”SOLVE:LINEAR_ELASTICITY:ISOTROPIC” を用いる場合の入力例である。

- bifurcation_plan：...動的条件分岐型シミュレーションの記述
  計算の一定時間または一定 STEP 毎に、条件分岐評価コマンドを計算し、その返り値により時間発展プロシージャを変えるしくみである。動的条件分岐の評価コマンドの選択、分岐先のプロシージャの候補と評価コマンドの返り値の関係を記述する。

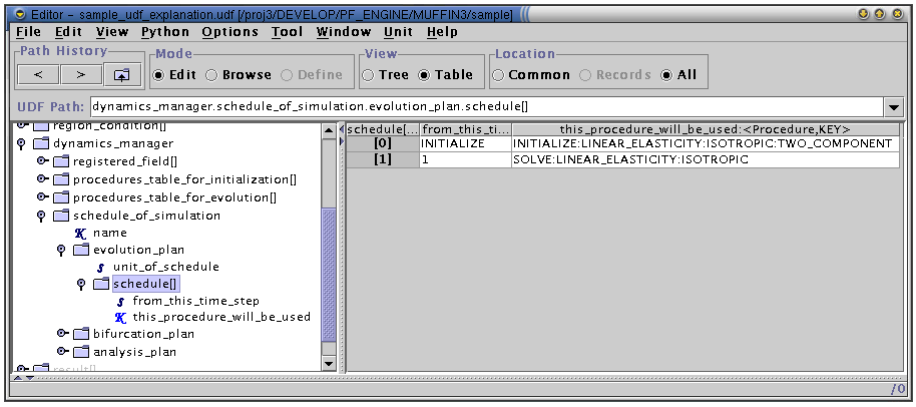


図 3.28: ダイナミクスマネージャ - 時間発展スケジュール - (UDF 解説)

– bifurcation_plan.unit_of_schedule : . . . タイムスケジュールに用いる単位
“TIME” または”STEP” のいずれかを入力する。

1. “TIME” . . . 時刻でスケジューリングを行う場合
2. “STEP” . . . 計算ステップでスケジューリングを行う場合

– bifurcation_plan.interval_for_evaluation : . . . 評価コマンド実行間隔
条件分岐評価コマンドを実行する間隔を記述する。bifurcation_plan.unit_of_schedule が”TIME” の場合には時間間隔 (double 型) を、“STEP” の場合には計算 STEP 間隔 (int 型) を記述する。

– bifurcation_plan.evaluation_function_name : . . . 条件分岐評価コマンド名
条件分岐評価コマンドを選択し、コマンド名を入力する。条件分岐評価コマンドの候補は、各シミュレータのリファレンスの” 場のコマンド一覧” のエーテル場 (Ether Field) の評価関数一覧を参照。
例えば、計算ステップで毎ステップ終了時に、”EVALUATE:LINEAR_ELASTICITY:ANISOTROPY” (弾性体の相分離構造が非等方であるかどうかを評価するコマンド) という条件分岐評価コマンドを実行する場合の入力例を、図 3.29 に示す。

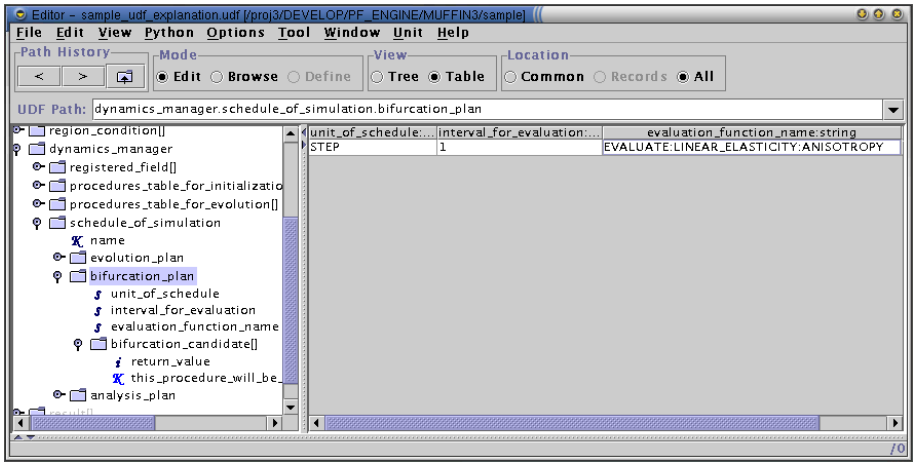


図 3.29: ダイナミクスマネージャ - 動的条件分岐・評価コマンド - (UDF 解説)

– bifurcation_plan.bifurcation_candidate[] : . . . 条件分岐先の候補プロシージャの記述
上で入力した条件分岐評価コマンドの返り値が 0 (評価結果が false) の場合、分岐は起らない。返り値が、$1, 2, 3, \ldots$ の自然数の場合のみ、時間発展プロシージャを変える。ここには、返り値の値と、入れ換える時間発展プロシージャの名前の関連を記述する。

* bifurcation_plan.bifurcation_candidate[].return_value：. . . 評価関数の返り値
  条件分岐評価関数の返り値を記述。返り値は、候補の先頭から 1, 2, 3, ... と自然数にする。
* bifurcation_plan.bifurcation_candidate[]. this_procedure_will_be_used_for_evolution_after_bifurcation：. . . 分岐先の時間発展プロシージャの名前
  各々の返り値に対応して、入れ換える時間発展プロシージャの名前を時間発展プロシージャテーブルから選択し記入する。

例えば、返り値が 1 の時、”SOLVE:LINEAR_ELASTICITY:ISOTROPIC” に時間発展プロシージャを入れ換え、2 の時、”SOLVE:LINEAR_ELASTICITY:ANISOTROPIC” に時間発展プロシージャを入れ換える場合の記入例を、図 3.30 に示す。

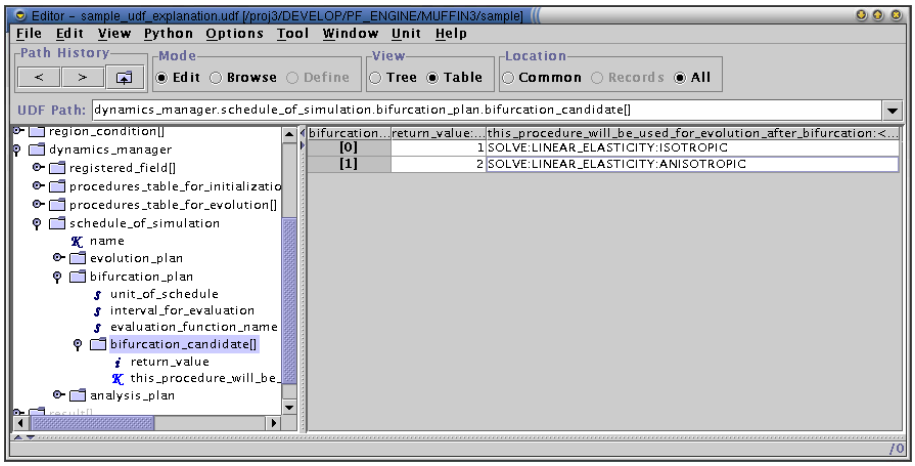


図 3.30: ダイナミクスマネージャ - 動的条件分岐・候補プロシージャ - (UDF 解説)

- analysis_plan：. . . 動的解析実行の記述
  計算の一定時間または一定 STEP 毎に、登録した場の、登録した解析実行評価コマンドを計算し、その返り値が、真 (非ゼロ) であれば、登録した解析実行コマンドを実行するしくみである。解析対象の場、解析評価コマンド、解析実行コマンドを記述する。動的解析実行の記述を行う。
  - analysis_plan.unit_of_schedule：. . . タイムスケジュールに用いる単位
    “TIME” または”STEP” のいずれかを入力する。
    1. “TIME” . . . 時刻でスケジューリングを行う場合
    2. “STEP” . . . 計算ステップでスケジューリングを行う場合
  - analysis_plan.interval_for_evaluation：. . . 評価コマンド実行間隔
    解析実行評価コマンドを実行する間隔を記述する。analysis_plan.unit_of_schedule が”TIME” の場合には時間間隔 (double 型) を、“STEP” の場合には計算 STEP 間隔 (int 型) を記述する。
    例えば、計算ステップで毎ステップ終了時に解析実行評価コマンドを実行する場合の入力例を、図 3.31 に示す。
  - analysis_plan.analyzer_list[]：. . . 解析する場、解析評価コマンド、解析実行コマンドのリスト
    解析対象の場、実際に解析を実行するかどうか動的に判定するための解析実行評価コマンド、解析実行評価コマンドの返り値が真 (非ゼロ) であった場合に実行する解析実行コマンドの 3 つの組み合わせのリストを登録する。場は登録する場のリストから選択する。各、場の評価コマンドの候補や解析実行コマンドの候補は、マニュアルの各シミュレータのリファレンス” 場のコマンド一覧” の解析対象の場の評価関数一覧と解析関数一覧より選択し入力する。
    * analysis_plan.analyzer_list[].field：. . . 解析対象の場の名前

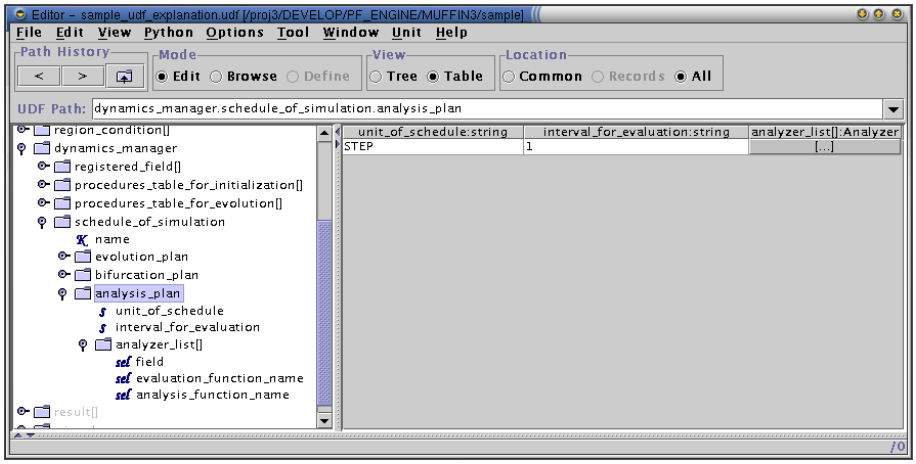


図 3.31: ダイナミクスマネージャ - 動的解析実行・評価コマンド - (UDF 解説)

* analysis_plan.analyzer_list[].evaluation_function_name：...場の解析評価コマンドの名前
* analysis_plan.analyzer_list[].analysis_function_name：...場の解析実行コマンドの名前

例えば、” 自由エネルギー場 (FreeEnergy)” と” 体積分率場 (VolumeFraction)” について、“EVALUATE:TRUE”(常に真を返す評価コマンド) で解析実行評価を行い。具体的な解析内容がいずれの場についても”OUTPUT:AVS”(AVS 形式で場のデータをファイル出力するコマンド) の場合の入力例を、図 3.32 に示す。

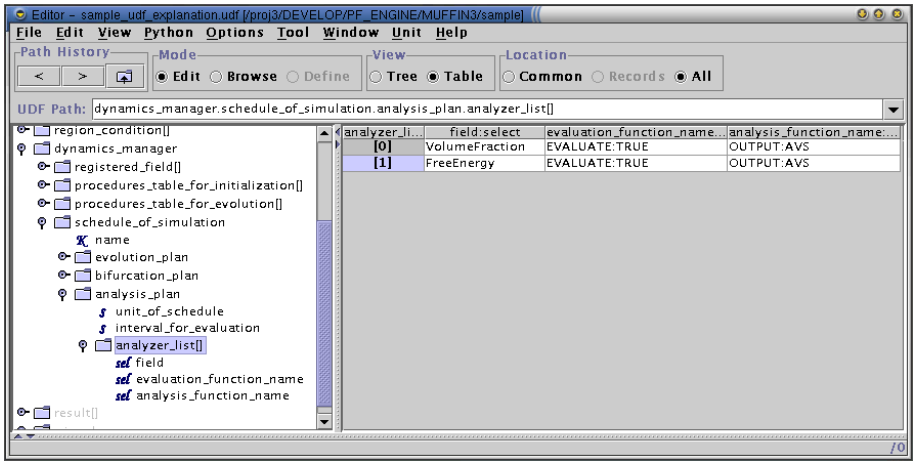


図 3.32: ダイナミクスマネージャ - 動的解析実行・解析実行コマンド - (UDF 解説)

3.3　コントロールパラメータ入力 UDF 解説

ここでは、Muffin エンジンの計算途中でのパラメータ変更をリアルタイムで行なうためのコントロールパラメータ入力 UDF (-P parameterUDF で指定) のデータ構造について解説する。

3.3.1　コントロールパラメータ入力 UDF 概要

コントロールパラメータ入力 UDF は、以下のように 2 つの大きなパートで構成される。

1. コントロールパラメータ部 (parameter[])
   計算途中で変更したいパラメータの名前と変更値を記述する。
2. 解析結果出力制御部 (summary_control)
   解析結果リアルタイム出力ファイル (summaryUDF) にレポートまたはグラフで出力するデータを指定する。
   - レポート出力データの制御 (report_result[])
   - グラフ出力データの制御 (graph_result[])

3.3.2 コントロールパラメータ部 (parameter[])

コントロールパラメータ部には、計算途中で変更したいパラメータのリストを入力する。各々のコントロールパラメータは、以下のデータ構造で入力する。

- name：...変更するパラメータの名前
  既存のパラメータならその値を更新する。もし、新規のパラメータが入力された場合には、そのパラメータを登録する。
- value[]：...変更する値 (配列)
  MUFFIN は、1 つの名前のパラメータに、複数の値を配列で与えることが出来る。

3.3.3 解析結果出力制御部 (summary_control)

次のセクションで解説する、解析結果リアルタイム出力ファイル (summaryUDF) にレポートまたはグラフで出力するデータを制御する。

- report_result[]：...レポート出力する解析結果データ名のリスト
  - name：...レポート出力する解析結果データ名
    レポート出力対象となるデータは、解析結果データ (出力 UDF の解析結果データ result[] に出力されるデータ) または入力パラメータ (入力 UDF のソルバパラメータ、共通物理パラメータ、物理パラメータ) である。入力パラメータのレポートは、パラメータのスケジューリングを行なっている場合に、そのパラメータが正常に変更されているかを確認する場合に有用である。
- graph_result[]：...グラフ出力する解析結果データのリスト
  - name：...グラフ出力する解析結果データ名
    レポート出力対象となるデータは、解析結果データ (出力 UDF の解析結果データ result[] に出力されるデータ) または入力パラメータ (入力 UDF のソルバパラメータ、共通物理パラメータ、物理パラメータ) である。入力パラメータのレポートは、パラメータのスケジューリングを行なっている場合に、そのパラメータが正常に変更されているかを確認する場合に有用である。
  - mode：...グラフ表示範囲のモード (”auto_range” or ”manual_range”)
    データ系列の最大値・最小値より自動的にレンジを決定する場合には ”auto_range” を、範囲を指定する場合には ”manual_range” を選択する。何も選択されなかった場合には、”auto_range” とみなされる。
  - manual_range：...グラフ表示範囲のモードが ”manual_range” の場合の範囲指定
    - min：...グラフ表示での軸の最小値
    - max：...グラフ表示での軸の最大値

3.4 解析結果リアルタイム出力 UDF 解説

ここでは、解析結果リアルタイム出力ファイル (-S summaryUDF で指定) のデータ構造を解説する。このデータ構造は、GOURMET のエンジンコントロールでの表示に対応するために、基本的にどのエンジンでも共通であり、簡単に解説する。

Muffin では、基本的にコントロールパラメータ入力 UDF で指定されたレポートデータやグラフデータがあればこれを出力する。

コントロールパラメータ入力 UDF の該当データが空である場合や、コントロールパラメータ入力 UDF 自体が-P オプション引数で指定されなかった場合 (例えば、バッチ処理で-S オプションを指定した場合) には、すべての解析結果データを出力する。この場合、入力パラメータは出力対象とならない。

- report_attribute : ... レポートデータの属性
  - location[] : ... レポートデータの UDF ロケーションの配列
  - label[] : ... レポートデータのラベルの配列
- graph_attribute : ... グラフデータの属性
  - title : ... グラフのタイトル
  - location[] : ... グラフデータの UDF ロケーションの配列
  - label[] : ... グラフデータのラベルの配列
  - scale[] : ... グラフ表示での軸の範囲
    - min : ... グラフ表示での軸の最小値
    - max : ... グラフ表示での軸の最大値
- report_data : ... レポートデータ
  - value[] : ... レポートデータの値の配列
- graph_data : ... グラフデータ
  - item[] : ... グラフデータのアイテムの配列
    - value[] : ... あるアイテムのグラフデータの時系列配列

第4章　Milk – 非構造格子生成ツール –

MILK は、UDF 形式の非構造格子を生成するプログラムである。非構造格子の生成には、デローネ 3 次元分割、および、デローネ 2 次元分割を利用している。

4.0.1　外部形状の生成

サポートしているメッシュ形状は、メッシュパラメータのメッシュのタイプ (parameter.mesh_parameter.type_of_mesh) が、

- **UNSTRUCTURED_RECT** . . . 非構造メッシュ(四角形、直方体 形状)
- **UNSTRUCTURED_SPHERE** . . . 非構造メッシュ(円状、球状)

の何れかである場合である。

メッシュパラメータ

各々の場合、メッシュの軸 axes[]、周期性 periodic[] は以下の意味である。

- **UNSTRUCTURED_RECT**
  - axes[0] . . . X 軸, axes[1] . . . Y 軸, axes[2] . . . Z 軸 (3 次元の場合)
  - periodic[0] . . . X 軸周期性, periodic[1] . . . Y 軸周期性, periodic[2] . . . Z 軸周期性 (3 次元の場合)
- **UNSTRUCTURED_SPHERE**
  - axes[0] . . . r 軸, axes[1] . . . θ 軸, axes[2] . . . ϕ 軸 (3 次元の場合)
  - periodic[0],periodic[1],periodic[2] . . . 周期性は常に 0 (false)

生成される部分領域

各々の場合に、生成される部分領域 (メッシュ要素の集合体) は以下の通りである。

- **UNSTRUCTURED_RECT**,**UNSTRUCTURED_SPHERE** 共通
  - “ALL_[VERTEX,EDGE,FACE,CELL]” . . . すべての [節点, 辺, 面, セル]
  - “ALL_BOUNDARY_[VERTEX,EDGE,FACE,CELL]”
    . . . すべての [境界節点, 境界辺, 境界面, 境界セル]
- **UNSTRUCTURED_RECT** のみ
  - “BOUNDARY_VERTEX_[XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX]”
    . . . [X 軸に垂直な底面の節点 (X 軸非周期), X 軸に垂直な上面の節点 (X 軸非周期),
    Y 軸に垂直な底面の節点 (Y 軸非周期), Y 軸に垂直な上面の節点 (Y 軸非周期),
    Z 軸に垂直な底面の節点 (Z 軸非周期), Z 軸に垂直な上面の節点 (Z 軸非周期)]

  - “BOUNDARY_FACE_[XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX]”
    . . . [X 軸に垂直な底面の面 (X 軸非周期), X 軸に垂直な上面の面 (X 軸非周期),
    Y 軸に垂直な底面の面 (Y 軸非周期), Y 軸に垂直な上面の面 (Y 軸非周期),
    Z 軸に垂直な底面の面 (Z 軸非周期), Z 軸に垂直な上面の面 (Z 軸非周期)]

- **UNSTRUCTURED_SPHERE** のみ
  - “CENTER_VERTEX” . . . 中心の節点 (内球面が無い場合)
  - “BOUNDARY_VERTEX_[TOP,BOTTOM]_CENTER” . . . [上, 底] 面上の中心点
  - “BOUNDARY_VERTEX_[OUTER,INNER]_SPHERE” . . . [外, 内] 球面上の節点
  - “BOUNDARY_VERTEX_[TOP,BOTTOM]” . . . [上, 底] 面上の節点
  - “BOUNDARY_VERTEX_[TOP,BOTTOM]_[OUTER,INNER]_SPHERE” . . . [外, 内] 球面と [上, 底] 面上の交線上の節点
  - “BOUNDARY_FACE_[OUTER,INNER]_SPHERE” . . . [外, 内] 球面上の面
  - “BOUNDARY_FACE_[TOP,BOTTOM]” . . . [上, 底] 面上の面

4.0.2 内部構造の生成

また、MILK は、上記の形状の内部に、以下の内部構造を複数持たせることができる。

- **SPHERE** . . . 3 次元球状
- **CIRCLE** . . . 2 次元円状
- **RECT** . . . 3 次元では直方体、2 次元では四角形

これら、内部構造の内側は、空洞にもできるし、内部をメッシュで満たすこともできる。

この仕組みにより、PhaseSeparation_FEM, Electrolyte_FEM を用いて、内部に球状や直方体の構造物を複数持つ (電解質) 流体シミュレーション、Elastica, GelDyna を用いて、空洞を複数持つ発泡体や中空円筒ゴムやゲルの弾性シミュレーションなどができる。

図 4.1 に、直方体の外部形状の内部に球状の構造物を入れたメッシュを示す。

内部構造パラメータ

内部構造の情報はソルバーパラメータか物理パラメータにより与える。以下に、パラメータによる内部構造の与え方について解説する。

パラメータの名前	パラメータの意味
INTERNAL_STRUCTURE	“1”(内部構造を与える) or “0”(内部構造は無い)
NUM_OF_INTERNAL_STRUCTURE	内部構造体の数 (0 以上の整数)
INTERNAL_STRUCTURE_num	各々の内部構造体のパラメータ配列 ($num = 0, 1, 2, ...NUM\_OF\_INTERNAL\_STRUCTURE - 1$)

以下に、パラメータ配列 INTERNAL_STRUCTURE_num の与え方を解説する。

- value[0] . . . “RECT” or “SPHERE” or “CIRCLE” (先頭データは、内部構造の形)
- value[1] . . . “1”(内部もメッシュで満たす) or “0”(内部は空洞)
  - “RECT”(3D) の場合

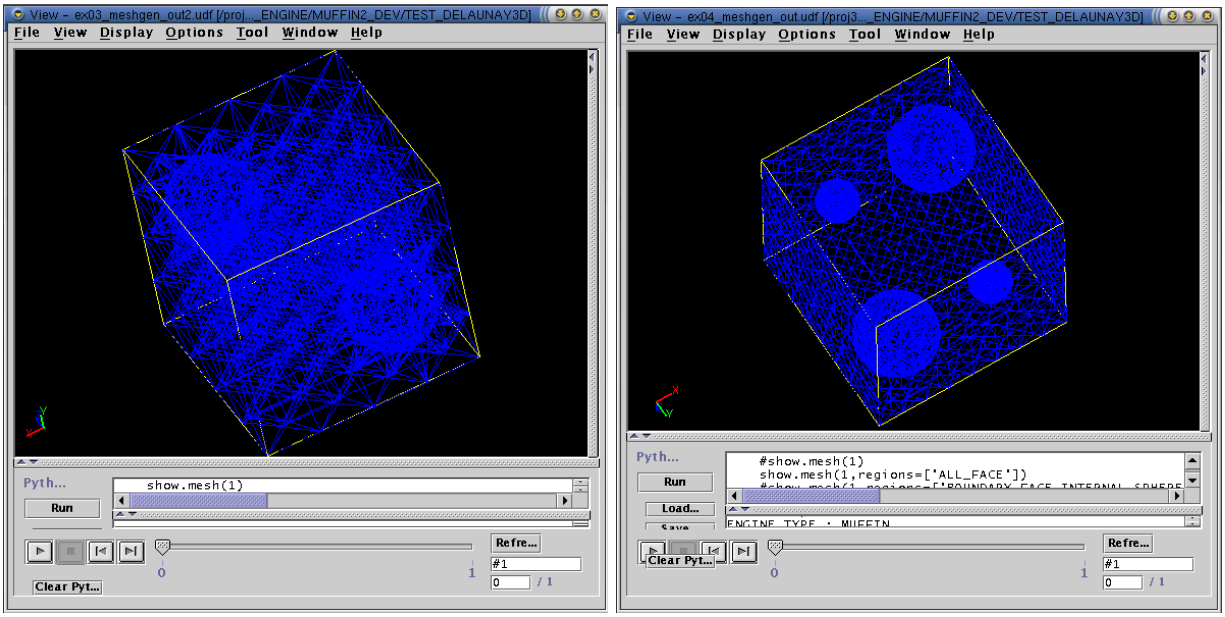


図 4.1: 直方体外部形状の非構造格子に中が空の球構造を入れたメッシュ。2 つの球を入れた場合の全四面体セル (左)、4 つの球を入れた場合の内部および表面の境界三角形 (右)。

    - value[2] ... 領域の X 座標最小値, value[3] ... Y 座標最小値, value[4] ... Z 座標最小値
    - value[5] ... 領域の X 座標最大値, value[6] ... Y 座標最大値, value[7] ... Z 座標最大値
    - value[8] ... 領域の X 座標の分割数, value[9] ... Y 座標の分割数, value[10] ... Z 座標の分割数
  - “RECT” (2D) の場合
    - value[2] ... 領域の X 座標の最小値, value[3] ... Y 座標の最小値
    - value[5] ... 領域の X 座標の最大値, value[6] ... Y 座標の最大値
    - value[8] ... 領域の X 座標の分割数, value[9] ... Y 座標の分割数
  - “SPHERE” の場合
    - value[2] ... 中心の X 座標, value[3] ... 中心の Y 座標, value[4] ... 中心の Z 座標
    - value[5] ... 球の半径
    - value[6] ... 半径 (r) の分割数 (空洞の場合には無視される)
    - value[7] ... 中心での $theta(0 \sim \pi)$ の分割数, value[8] ... 球の極での phi の分割数
  - “CIRCLE” の場合
    - value[2] ... 中心の X 座標, value[3] ... 中心の Y 座標
    - value[5] ... 円の半径
    - value[6] ... 半径 (r) の分割数 (空洞の場合には無視される)
    - value[7] ... 中心での $theta(0 \sim 2\pi)$ の分割数

生成される内部構造の部分領域

内部構造の内部や表面につけられる部分領域の名前は、内部構造体の識別のために、上記の “INTERNAL_STRUCTURE_num” の文字列を含む。下記の解説では、“INTERNAL_STRUCTURE_num” の識別名の部分を “XXXX” として解説する。

また、

- “ALL_[VERTEX,EDGE,FACE,CELL]_XXXX” . . . 内部構造体に含まれるすべての要素は、”ALL_[VERTEX,EDGE,FACE,CELL]” にも含まれる。
- “ALL_BOUNDARY_[VERTEX,EDGE,FACE]_XXXX” . . . 内部構造体のすべての境界要素は、”ALL_BOUNDARY_[VERTEX,EDGE,FACE]” にも含まれる。

ことに注意。

- “RECT”,“SPHERE” および “CIRCLE” 共通
  - “ALL_VERTEX_XXXX” . . . 内部構造体に含まれるすべての節点
- “RECT” の場合のみ
  - “ALL_CELL_XXXX” . . . 内部構造体に含まれるすべてのセル
  - “BOUNDARY_VERTEX_[XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX]_XXXX” . . . 内部構造体の [X 軸に垂直な底面,X 軸に垂直な上面, Y 軸に垂直な底面,Y 軸に垂直な上面, Z 軸に垂直な底面,Z 軸に垂直な上面] の節点
  - “BOUNDARY_FACE_[XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX]_XXXX” . . . 内部構造体の [X 軸に垂直な底面,X 軸に垂直な上面, Y 軸に垂直な底面,Y 軸に垂直な上面, Z 軸に垂直な底面,Z 軸に垂直な上面] の面
- “SPHERE” の場合のみ
  - “ALL_CELL_XXXX” . . . 内部構造体に含まれるすべてのセル
  - “CENTER_VERTEX_XXXX” . . . 内部構造体の中心の節点
  - “BOUNDARY_VERTEX_XXXX” . . . 内部構造体の球面上の節点
  - “BOUNDARY_VERTEX_XXXX_num” . . . 内部構造体の球面上の部分領域の面 $num = 00, 01, 02, 03, 10, 11, 12, 13$ が各々、第 1 象限, 第 2 象限, 第 3 象限, 第 4 象限, 第 5 象限, 第 6 象限, 第 7 象限, 第 8 象限を表す。
  - “BOUNDARY_FACE_XXXX” . . . 内部構造体の球面上の面
  - “BOUNDARY_FACE_XXXX_num” . . . 内部構造体の球面上の部分領域の面 $num = 00, 01, 02, 03, 10, 11, 12, 13$ が各々、第 1 象限, 第 2 象限, 第 3 象限, 第 4 象限, 第 5 象限, 第 6 象限, 第 7 象限, 第 8 象限を表す。
- “CIRCLE” の場合のみ
  - “ALL_FACE_XXXX” . . . 内部構造体に含まれるすべての面
  - “CENTER_VERTEX_XXXX” . . . 内部構造体の中心軸上の節点
  - “BOUNDARY_VERTEX_XXXX” . . . 内部構造体の面上の節点
  - “BOUNDARY_EDGE_XXXX” . . . 内部構造体の面上の辺

4.0.3 生成したメッシュのコピー

このプログラムにより生成したメッシュは、アクション ”export_mesh” プログラムにより他の UDF ファイルにコピーすることができるので、これにより、入力 UDF ファイルを作成し、メッシュのタイプ (parameter.mesh_parameter.type_of_mesh) を、

- UNSTRUCTURED_INPUT . . . 非構造メッシュデータ入力

としてメッシュデータを読み込み、シミュレーションを行うことが出来る。

詳細は、”5.3 共通 UDF/コンバートプログラム” を参照。

第5章　プリプロセッサ

5.1　Modeler – Elastica, Milk 用モデラー –

Elastica と Milk のためのモデラーを UDF ファイルの”modeler.Elastica”および”modeler.Milk”部に用意した。入力後、アクション”run_modeler”, ”run_elastica_modeler”,... などにより入力ファイルの作成が実行される。また、”run_modeler_and_engine”, ”run_elastica_modeler_and_engine”, ”run_milk_modeler_and_engine”, ... などにより入力ファイルの作成と保存、エンジン実行、結果 UDF ファイルの Open までが実行される。

場とメッシュを他の UDF ファイルから入力する場合にはアクション”import_field”を用いて import しておき、モデラーではモルフォロジーを”import”に指定しておく。また、メッシュのみを Milk の出力結果などの他の UDF ファイルから入力する場合にはアクション”import_mesh”を用いてメッシュデータを import しておき、モデラーでは mesh を”input”に指定しておくと良い。

5.2　MuffinMujigen – 入力パラメータ無次元化ツール –

MUFFUN では、PhaseSeparation シミュレータで用いられる、各パラメータの無次元化を行うプログラムを用意している。パラメータの無次元化の詳細については、PhaseSeparation の理論背景部などを参考にして欲しい。ここでは、無次元化プログラムの使い方について、説明する。

5.2.1　無次元化プログラムの機能解説

無次元化プログラムは、MKSA 単位系でパラメータが記述された入力 UDF ファイルを読み込み、すべてのパラメータを無次元化し変換した出力 UDF ファイルを生成するプログラムである。出力 UDF ファイルは入力 UDF ファイルがコピーされ、更に、パラメータ部分が変換されるので、入力の元データが壊されることも無い。

処理の流れの概要は以下の通り。

1. 入力 UDF ファイルから基本パラメータを読み取り、電解質シミュレーションの有無を判断。
2. 基本パラメータから無次元化単位を計算。
3. 入力 UDF ファイルをコピーし、出力 UDF ファイルを作成。
4. 無次元化変換対象のすべてのパラメータが無次元化され出力される。

無次元化の基本パラメータ

無次元化プログラムでは、入力 UDF ファイルに、次の基本パラメータが単位つきの実際の値で必須である。これらは、入力 UDF ファイルの物理パラメータ部、または、ソルバパラメータ部に登録されていなくてはならない。

電解質シミュレーション Electrolyte であるか、多相構造シミュレーション PhaseSeparation であるかは、“AVERAGED_ION_CONCENTRATION” パラメータと、“AVERAGED_VOLUME_FRACTION” パラメー

タの有無で判断される。両方のパラメータがある場合には、“AVERAGED_VOLUME_FRACTION” パラメータが優先され、多相構造シミュレーションと見なされる。両方ともパラメータが無い場合には、エラーとなる。

基本パラメータとは以下のパラメータである。(ここでは、パラメータ名だけで説明する。各々のパラメータの意味は、PhaseSeparation リファレンスを参照。)

1. 多相構造シミュレーションの基本パラメータ
   - MONOMER_SIZE ... モノマーのサイズ (nm)
   - NUMBER_OF_COMPONENTS ... 成分数
   - AVERAGED_VOLUME_FRACTION ... 各成分の仕込み体積分率 (配列)
   - POLYMERIZATION_INDEX_N ... モノマーを単位とした各成分の重合度 (配列)
   - CHI_mn ... 成分 m と n の χ パラメータ
   - VISCOSITY ... 各成分の粘度 (Pa·sec)(配列)
   - DIFFUSION_COEFFICIENT ... 各成分の拡散定数 (cm^2/sec)(配列)
   - TEMPERATURE ... 温度 (K)
2. 電解質シミュレーションの基本パラメータ
   - NUMBER_OF_COMPONENTS ... 成分数
   - AVERAGED_ION_CONCENTRATION ... 各成分の仕込み濃度 (mol/l)(配列)
   - DIFFUSION_COEFFICIENT ... 各成分の拡散定数 (cm^2/sec)(配列)
   - Z ... 各成分のイオンの荷数 (配列)
   - TEMPERATURE ... 温度 (K)

無次元化プログラムのインターフェース

使い方は、

% MuffinMujigen.py input_dim.udf output_nondim.udf
である。

5.2.2　自動変換されるパラメータと次元

以下に、無次元化変換対象のすべてのパラメータをその次元毎に述べる。これらのパラメータは、入力 UDF ファイルに存在すれば、自動的に変換されて出力 UDF ファイルの同じデータ項目に出力される。

濃度勾配 (mol/l/nm) の次元を持つパラメーター覧

これらは、電解質シミュレーションのパラメータである。
CONCENTRATION_GRADIENT_ALONG_X, CONCENTRATION_GRADIENT_ALONG_Y, CONCENTRATION_GRADIENT_ALONG_Z

長さの次元 (nm) を持つパラメーター覧

RADIUS_OF_DROPLET

拡散定数の次元 ($\mathbf{cm^2/sec}$) を持つパラメーター一覧

DIFFUSION_COEFFICIENT

体積力の次元 ($\mathbf{g/sec^2/nm^2}$) を持つパラメーター一覧

GRAVITY_X, GRAVITY_Y, GRAVITY_Z

電場ポテンシャルの次元 ($\mathbf{kV}$) を持つパラメーター一覧

ELECTRIC_POTENTIAL_AT_YZ_PLANE_XM, ELECTRIC_POTENTIAL_AT_YZ_PLANE_XP,
ELECTRIC_POTENTIAL_AT_ZX_PLANE_YM, ELECTRIC_POTENTIAL_AT_ZX_PLANE_YP,
ELECTRIC_POTENTIAL_AT_XY_PLANE_ZM, ELECTRIC_POTENTIAL_AT_XY_PLANE_ZP,
CONSTANT_TERM_OF_ELECTRIC_POTENTIAL_OSCILLATION_AT_YZ_PLANE_XP,
AMPLITUDE_OF_ELECTRIC_POTENTIAL_OSCILLATION_AT_YZ_PLANE_XP

電場 ($\mathbf{kV/mm}$) の次元を持つパラメーター一覧

ELECTRIC_POTENTIAL_GRADIENT, GRADIENT_OF_E-Potential_AT_YZ_PLANE_XM,
GRADIENT_OF_E-Potential_AT_YZ_PLANE_XP, GRADIENT_OF_E-Potential_AT_ZX_PLANE_YM,
GRADIENT_OF_E-Potential_AT_ZX_PLANE_YP, GRADIENT_OF_E-Potential_AT_XY_PLANE_ZM,
GRADIENT_OF_E-Potential_AT_XY_PLANE_ZP

角速度 ($\mathbf{rad/sec}$) の次元を持つパラメーター一覧

FREQUENCY_OF_ELECTRIC_POTENTIAL_OSCILLATION_AT_YZ_PLANE_XP,
FREQUENCY_OF_P_OSCILLATION_AT_YZ_PLANE_XP

速度 ($\mathbf{m/sec}$) の次元を持つパラメーター一覧

VX_AT_YZ_PLANE_XM, VY_AT_YZ_PLANE_XM, VZ_AT_YZ_PLANE_XM,
VX_AT_YZ_PLANE_XP, VY_AT_YZ_PLANE_XP, VZ_AT_YZ_PLANE_XP,
VX_AT_ZX_PLANE_YM, VY_AT_ZX_PLANE_YM, VZ_AT_ZX_PLANE_YM,
VX_AT_ZX_PLANE_YP, VY_AT_ZX_PLANE_YP, VZ_AT_ZX_PLANE_YP,
VX_AT_XY_PLANE_ZM, VY_AT_XY_PLANE_ZM, VZ_AT_XY_PLANE_ZM,
VX_AT_XY_PLANE_ZP, VY_AT_XY_PLANE_ZP, VZ_AT_XY_PLANE_ZP

表面電荷 ($\mathbf{C/m^2}$) の次元を持つパラメーター一覧

SURFACE_CHARGE_ON_OBSTACLE

電荷密度の次元 ($\mathbf{C/m^3}$) を持つパラメーター一覧

CHARGE_DENSITY

圧力の次元 (Pa) を持つパラメーター一覧

PRESSURE_GRADIENT, PRESSURE_AT_YZ_PLANE_XM, PRESSURE_AT_YZ_PLANE_XP, PRESSURE_AT_ZX_PLANE_YM, PRESSURE_AT_ZX_PLANE_YP, PRESSURE_AT_XY_PLANE_ZM, PRESSURE_AT_XY_PLANE_ZP, CONSTANT_VALUE_OF_P_OSCILLATION_AT_YZ_PLANE_XP, AMPLITUDE_OF_P_OSCILLATION_AT_YZ_PLANE_XP

その他、電解質シミュレーションでの無次元化対象パラメータ

- R . . . 静電エネルギーと熱エネルギーの比,
- M
- D

その他、多相構造シミュレーションでの無次元化対象パラメータ

- B . . . 電気エネルギー
- DIELECTRIC_CONSTANT . . . 誘電率 ($C^2/N/m^2$)
- CA . . . キャピラリー数
- VISCOSITY . . . 粘度 (Pa·sec),
- SHEAR_RATE_XZ . . . シアレート (1/sec)

5.3 MeshFieldConvertor – 場とメッシュのコンバートツール –

複数のメッシュと場と共通 UDF を扱う UDF ファイル間での解析やデータコンバートなどを行うプログラムである。現バージョンでは、SUSHI から MUFFIN、MUFFIN から SUSHI、MUFFIN 間、SUSHI 間のメッシュや場のデータコンバート機能を持つ。

5.3.1 MeshFieldConvertor の機能

1. ヘッダーデータでエンジンタイプを自動判定
2. 用いているメッシュのタイプを自動判定
3. UDF にある場のデータはすべて自動検索
4. 各エンジン (現在は SUSHI,MUFFIN) で使用している場の名称のシソーラス (辞書) を持ち、SUSHI の phi が、MUFFIN の VolumeFraction に対応することなど自動認識
5. コンバートではシソーラスにより対応するすべての場のデータを両 UDF のメッシュタイプに応じて自動コンバート。
6. メッシュデータのコンバート機能

5.3.2 MeshFieldConvertor のインターフェース

Shell から Python プログラムを直接実行する場合は、コンバート元とコンバート先の UDF ファイル名とレコード番号を更に実行する処理を引数に指定して実行する。Usage は以下の通り。

% MeshFieldConvertor.py -i *inputUDF* { -ir *input_record_No.* } -o *outputUDF* { -or *output_record_No.* } { -e *execute_program* }

- inputUDF . . . コンバート元の UDF ファイルのパス
- input_record_No. . . . コンバート元の UDF データのレコード番号 (デフォルト -1)
- outputUDF . . . コンバート先の UDF ファイルのパス
- output_record_No. . . . コンバート先の UDF データのレコード番号 (デフォルト -1)
- execute_program . . . 実行するプログラム。(デフォルト convert)
  現在、-e convert と -e convert_mesh が使用可能。

5.3.3 SUSHI,MUFFIN など異なる UDF 間の場のデータのズームインとズームアウト

SUSHI の出力 UDF からメッシュパラメータと場のデータを読み、MUFFIN の入力 UDF を構築する方法を解説する。

1. GOURMET の Java Python での使い方
   Java Python では、GOURMET で現在開いている UDF ファイルの該当レコードに異なる UDF ファイルのあるレコードのデータをコンバートすることを想定している。
   MeshFieldConvertor.py の変換元のファイルパスを編集して使用する。

```
....
import muffinlib.MultiMeshFieldAnalysisLib
# コンバート元の UDF ファイル名
import_udf_path="/home/yamaue/...../cylinder3D_b_uot.udf"
# コンバート元のレコード番号 ( -1 は common データを指す )
import_udf_record=1
....
```

2. Shell から Python プログラムを直接実行する場合
   コンバート元とコンバート先の UDF ファイル名とレコード番号を更に実行する処理を引数に指定して実行する。Usage は上記の通り。

5.3.4 アクション操作によるズームインとズームアウト

アクション操作による SUSHI から MUFFIN へのズームアウト

場のデータの UDF パス（field）を右クリックすると、”zoom_out_from_sushi()” メニューが出るので、これを実行することで、SUSHI から MUFFIN への場のデータのコンバートが出来る。実行時に変換元の SUSHI の UDF ファイルのパスとレコード番号を入力するダイアログが開く。

アクション操作による MUFFIN から SUSHI へのズームイン

場のデータの UDF パス（field）を右クリックすると、”zoom_in_to_sushi()” メニューが出るので、これを実行することで、MUFFIN から SUSHI への場のデータのコンバートが出来る。実行時に変換先の SUSHI の UDF ファイルのパスとレコード番号を入力するダイアログが開く。

5.4 NASTRAN BULK ファイルコンバートツール

ここでは、NASTRAN BULK ファイル形式のメッシュデータを共通 UDF 形式のメッシュデータに変換する方法を解説する。

操作の手順は大きく分けて、以下の 2 ステップである。

1. コンバートプログラムを動かし、共通 UDF 形式のメッシュデータを作成する。
2. 共通 UDF 形式のメッシュデータについて、部分領域作成の python プログラム (MakePartialRegions.py) を実行する。

コンソールで変換の python プログラムを実行する場合には、以上の 2 ステップは一括で実行される。

5.4.1 GOURMET 上でのコンバートプログラム実行

GOURMET 上での共通 UDF 形式へのコンバート

1. “File” メニューの”ConvertFile...” をクリックし、”ConvertFile” ダイアログを開く。
2. “Data File:” に、変換元の NASTRAN BULK ファイルを選択する。
3. “Rule File:” に、NASTRAN BULF 形式 -¿ 共通 UDF 形式のフィルタである “$PF_ENGINE/python/nastran/nastranbull を選択する。
4. “Input UDF:” に、MUFFIN3 の UDF 定義部である “$PF_ENGINE/def_udf/muffin3.udf” を選択する。
5. “Output UDF:” に、変換先の UDF ファイル名を入力する。
6. “OK” を押し、変換を実行する。

GOURMET 上での部分領域作成プログラムの実行

1. GOURMET で、変換した共通 UDF 形式のメッシュデータを開く。
2. 部分領域作成のための python プログラム”MakePartialRegions.py” を開き (“Load”)、実行 (“Run”) する。
3. 部分領域名に適切な名前を付けるなど編集し、保存する。

5.4.2 コンソールでのコンバートプログラム実行

コンソールを開き、NASTRAN BULF 形式 -¿ 共通 UDF 形式の変換 python プログラム “nasbulk2udf.py” を実行する。
Usage: nasbulk2udf.py bulkdata.dat outdata.udf [def.udf]

ここで、

- bulkdata.dat . . . 変換元の NASTRAN BULK ファイルのパス。
- outdata.udf . . . 変換先の UDF ファイルのパス。
- def.udf . . . オプション：変換先の UDF データの定義部
  指定が無い場合、デフォルトの muffin3.udf となる。

変換プログラム実行後、UDF 形式のメッシュデータを GOURMET で開き、部分領域名に適切な名前を付けるなど編集し、保存する。

第6章　ポストプロセッサ

6.1　MeshFieldPlot – 場のデータ解析・プロットツール –

MeshFieldPlot.py は、場のデータの空間分布や空間相関関数を解析するプログラムである。

ここで、共通 UDF の場の断面・表面・直線に沿った空間分布データを GOURMET の Java Python により解析し、GraphSheet の作成と gnuplot での Plot 表示を行う python プログラム MeshFieldPlot.py の機能と使い方の解説を行う。

SUSHI の空間相関関数解析プログラム (spcf) とのインターフェースも持ち、GOURMET 上で空間相関関数解析を行い GraphSheet の作成と gnuplot での Plot 表示を行うことも出来る。

6.1.1　MeshFieldPlot.py の機能解説

1. 場の断面の空間分布データの抽出と GraphSheet 作成
2. 場の表面の空間分布データの抽出と GraphSheet 作成
3. 場の線分に沿った空間分布データの抽出と GraphSheet 作成
4. SUSHI の spcf を用いた場の空間相関関数解析と GraphSheet 作成
5. スカラー場、ベクトル場など場のタイプは自動判定
6. ヘッダーデータによりエンジンタイプ (SUSI、MUFFIN) を判定し、各々のエンジンの UDF に対応。共通のインターフェースで利用可能

6.1.2　MeshFieldPlot.py のインターフェース解説

1. インターフェース

   (a) コンストラクタ：MeshFieldPlot()
   MeshFieldPlot オブジェクトを生成する。

   (b) メソッド：field(name,component = 0,region = ”,cplane=[],cline=[])
   場の断面・表面・線分に沿った串刺の空間分布データ解析し GraphSheet を生成する。
   引数解説

   - name . . . 場の名前 (必須)
   - component . . . 解析する場の成分の番号 (デフォルト値,0)
   - region . . . 解析する断面や表面の領域名 (デフォルト値,”)
     構造格子 (REGULAR,RECTANGULAR)、非構造格子のすべてで使用可能。
   - cplane . . . 解析する断面や (表面の) の指定 (デフォルト値,[])
     構造格子 (REGULAR,RECTANGULAR)、非構造格子 (UNSTRUCTURED_RECT) で使用可能。cplane=[[point on plane],[normal vector]] の形式で指定
     ex.) cplane=[[0,0,0],[0,0,1]] 原点を通り Z 軸に垂直な面 (XY 平面)

- cline . . . 串刺解析する線分の指定 (デフォルト値,[])
  構造格子 (REGULAR,RECTANGULAR)、非構造格子 (UNSTRUCTURED_RECT) で使用可能。cline=[[point on line],[direction vector]] の形式で指定
  ex.) cline=[[0,0,0],[0,0,1]] 原点を通り Z 軸に平行な線分 (Z 軸)

(c) メソッド : spcf(name,component = 0,region = ”,cplane=[],direction=[],minmax=[])
場の空間相関関数解析し SUSHI の spcf の入力 UDF ファイルを生成する。
引数解説

- name . . . 場の名前 (必須)
- component . . . 解析する場の成分の番号 (デフォルト値,0)
- region . . . 解析する断面や表面の領域名 (デフォルト値,”:すべての領域)
  構造格子 (REGULAR,RECTANGULAR)、非構造格子のすべてで使用可能。
- cplane . . . 解析する断面や (表面の) の指定 (デフォルト値,[]:すべての領域)
  構造格子 (REGULAR,RECTANGULAR)、非構造格子 (UNSTRUCTURED_RECT) で使用可能。cplane=[[point on plane],[normal vector]] の形式で指定
  ex.) cplane=[[0,0,0],[0,0,1]] 原点を通り Z 軸に垂直な面 (XY 平面)
- direction . . . 特定方向に関する相関関数を計算する場合の方向ベクトル
  デフォルトは [] でこの場合等方的な相関関数が計算される。
  ex.) direction=[0,0,1] Z 軸に平行な方向の相関を計算。
- minmax . . . データ・クリップを行う場合に下限値と上限値を与える。
  デフォルトは [] でクリップを行わない。

2. 使い方

(a) 場の断面・表面・線分に沿った串刺の空間分布データ解析
下記のプログラムを入力せずとも、MeshFieldPlot.py を Load し、最後にある main 関数の記述を編集して用いても良い。

```
# パッケージ MeshFieldPlot を import
from MeshFieldPlot import *
# MeshFieldPlot を生成
plot = MeshFieldPlot()
# 構造格子のスカラー場 'Concentration' を cplane 指定で断面解析
#plot.field('Concentration',cplane=[ [0.,0.,0.],[0.,0.,1.] ] )
# 非構造格子のスカラー場 'FreeEnergy' を領域指定で表面解析
plot.field('FreeEnergy',region='YMIN' )
# 非構造格子のベクトル場 'Displacement' を領域指定で表面解析
#plot.field('Displacement',region='ZMIN' )
# 非構造格子のスカラー場 'VolumeFraction' を線分指定で串刺解析
#plot.field('VolumeFraction',cline=[ [0.,0.,0.],[0.,0.,1.] ] )
```

(b) 場の空間相関関数解析
下記のプログラムを入力せずとも、MeshFieldPlot.py を Load し、最後にある main 関数の記述を編集して用いても良い。

```
# パッケージ MeshFieldPlot を import
```

```
from MeshFieldPlot import *
# MeshFieldPlot を生成
plot = MeshFieldPlot()
# 非構造格子のスカラー場 'FreeEnergy' を空間相関解析
plot.spcf('FreeEnergy')
```

6.1.3 アクション操作による Plot プログラムの実行

アクション操作による場のデータの Plot

各場のデータの UDF パス（field, field.scalar_field[], field.vector_field[]) を右クリックすると、"plot_xxxx()" メニューが出るので、これを実行することでも場のデータの Plot が行える。実行時に必要なパラメータを入力するダイアログが開く。

アクション操作による空間相関関数解析

場か、各スカラー場のデータの UDF パス（field, field.scalar_field[]) を右クリックすると、"spcf()" メニューが出るので、これを実行することでも場の空間相関関数解析が行える。実行時に必要なパラメータを入力するダイアログが開く。

6.2 MeshFieldShow – 場のコンター描画ツール –

共通 UDF のメッシュと場のデータを GOURMET の Java Python により描画する python プログラム MeshFieldShow.py の機能と使い方の解説を行う。

6.2.1 MeshFieldShow.py の機能解説

1. スカラー場の断面・表面のカラーコンターの描画
2. ベクトル場の断面・表面のベクトルの描画
3. スカラー場、ベクトル場など場のタイプは自動判定
4. ヘッダーデータによりエンジンタイプ (SUSHI、MUFFIN) を判定し、各々のエンジンの UDF に対応。共通のインターフェースで利用可能

6.2.2 MeshFieldShow.py のインターフェース解説

1. インターフェース
   (a) コンストラクタ：MeshFieldShow()
   MeshFieldShow オブジェクトを生成する。
   (b) メソッド：field(name,component=0,regions=[],cplanes=[],
   minmax=[0,1],color_att=1,arrow_att=[0,1,0,1,0,0], skip_att=0)
   断面や指定領域 (複数指定可能) のスカラー場のカラーコンター、ベクトル場のベクトルを描画する。引数解説

- name . . . 場の名前 (必須)
- component . . . 描画する場の成分の番号 (デフォルト値,0)
- regions . . . 描画する断面や表面の領域名 (デフォルト値,[])
  構造格子 (REGULAR,RECTANGULAR)、非構造格子のすべてで使用可能。
  regions=['region_name_1','region_name_2',...'region_name_n']
  複数の領域の指定可能。
  ex.) regions=['ZMIN','ZMID','ZMAX'] Z 軸に垂直な 3 枚の面で切る
- cplanes . . . 描画する断面や (表面の) の指定 (デフォルト値,[])
  構造格子 (REGULAR,RECTANGULAR)、非構造格子 (UNSTRUCTURED_RECT) で使用可能。cplanes=[[[point on plane],[normal vector]],...., [[point on plane],[normal vector]]] の形式で指定
  複数の断面の指定可能。
  ex.) cplanes=[[[0,0,0],[0,0,1]],[[0,0,0],[0,1,0]]] XY 平面と XZ 平面
- minmax . . . スカラー場の最小、最大値 (デフォルト,[0,1])
- color_att . . . スカラー場のカラー属性の指定 (デフォルト,1)
- arrow_att . . . ベクトル場のベクトル属性の指定
  arrow_att=[R(ED),G(REEN),B(LUE),T(RANSPARENCY), radius of arrowhead,length of arrowhead]
- skip_att . . . 構造格子での描画する頂点のスキップ数 (デフォルト,0)

(c) メソッド : mesh(color_att=1,regions=[])
断面や指定領域 (複数指定可能) のメッシュ格子を描画する。引数解説

- color_att . . . 格子を描画する線のカラー属性の指定 (デフォルト,1)
- regions . . . 格子を描画する断面や表面の領域名 (デフォルト値,['ALL_CELL'])
  構造格子 (REGULAR,RECTANGULAR)、非構造格子のすべてで使用可能。
  regions=['region_name_1','region_name_2',...'region_name_n']
  複数の領域の指定可能。
  ex.) regions=['ZMIN','ZMID','ZMAX'] Z 軸に垂直な 3 枚の面で切る

(d) メソッド : frame(color_att=0)
空間領域のフレームを描画
引数解説

- color_att . . . フレームを描画する線のカラー属性の指定 (デフォルト,1)

2. 使い方
下記のプログラムを入力せずとも、MeshFieldShow.py を Load し、最後にある main 関数の記述を編集して用いても良い。

```
# パッケージ MeshFieldShow を import
from MeshFieldShow import *
# MeshFieldShow を生成
show = MeshFieldShow()

# SUSHI の構造格子のスカラー場 'phi' を cplane 指定で描画
#show.field('phi',cplanes=[[[0.,0.,0.],[0,0,1]],[[0.,0.,0.],[0,1,0]]])
# MUFFIN の構造格子のスカラー場 'Concentration' を cplane 指定で描画
```

```
#show.field('Concentration',cplane=[ [0.,0.,0.],[0.,0.,1.] ] )
# 非構造格子のスカラー場 'VolumeFraction' を表面領域指定で描画
#show.field('VolumeFraction', regions=[ 'ALL_FACE' ])
# 非構造格子のベクトル場 'Displacement' を表面領域指定で描画
#show.field('Displacement',regions=[ 'ALL_FACE' ] )
# 非構造格子のスカラー場 'FreeEnergy' を表面と内部の断面指定で描画
#show.field('FreeEnergy',regions=['ZMIN','ZMID','ZMAX'],minmax=[0.0,1.5])
```

6.2.3 アクション操作による描画プログラムの実行

アクション操作による場のコンター描画

各場のデータの UDF パス（field, field.scalar_field[], field.vector_field[]) を右クリックすると、"show_xxxx()" メニューが出るので、これを実行することでも場のコンター描画が行える。実行時に必要なパラメータを入力するダイアログが開く。

アクション操作によるメッシュの描画

メッシュデータの UDF パス（mesh_coordinate, mesh_element) を右クリックすると、"show()" メニューが出るので、これを実行することでもメッシュの描画が行える。実行時に必要なパラメータを入力するダイアログが開く。

6.3 udf2avs – UDF 形式から AVS 形式への変換ツール –

共通 UDF に基づく MUFFIN エンジンの出力 UDF ファイルのメッシュと場のデータを AVS(Application Visualization System) で表示可能な形式に変換する Python プログラムである。

使用法

python muffin3_udf2avs.py -d *AVS-file-output-directory* *UDF-file*

実行時オプション "-d *AVS-file-output-directory*" によって AVS ファイルを出力するディレクトリを指定できる。このオプションを省略すると実行したディレクトリ以下の "AVS" というディレクトリが指定されたことになる。ディレクトリが存在しない場合には自動的に作成される。

生成されるファイル

ディレクトリ *AVS-file-output-directory* に以下のようなファイルがつくられる。

- **有限要素法シミュレータ (PhaseSeparation_FEM、Electrolyte_FEM, Reaction, Elastica, Ela-Dyna, GelDyna) の出力する UDF の場合:**

  UDF の各レコードに含まれる場の出力データごとに AVS の UCD(Unstructured Cell Data) 形式のファイルが以下のような名前でつくられる；

  <*Field-name*><*step-number*>.inp

ここで <*Field-name*> は Velocity、VolumeFraction 等の場の名前であり、<*step-number*> はデータが出力されたステップ番号であり “000100”、“012000” のような 6 桁の整数で出力される。

- **有限差分法シミュレータ (PhaseSeparation_FDM、Electrolyte_FDM) の出力する UDF の場合:**

  UDF の各レコードに含まれる場の出力データごとに AVS の field file 形式のファイルとデータ本体ファイルが以下のような名前でつくられる；

  <*Field-name*><*step-number*>`.fld`
  <*Field-name*><*step-number*>`.dat`

  `*.fld` は field file 形式の データで、`*.dat` はデータを記述したアスキー形式データファイルである。<*Field-name*> は Velocity、VolumeFraction 等の場の名前であり、<*step-number*> はデータが出力されたステップ番号であり “000100”、“012000” のような 6 桁の整数で出力される。

付 録A　コンパイル方法

A.1　ソースの再コンパイル

A.1.1　環境変数の設定

1. エンジンをインストールしたトップディレクトリ を PF_ENGINE に設定。
2. GOURMET をインストールしたトップディレクトリ を PF_FILES に設定。

A.1.2　コンパイル

unix 系、linux 系、cygwin での再コンパイル

エンジンをインストールしたディレクトリの下の MUFFIN のソースディレクトリに移動し、Makefile を用いて、コンパイル します。

% cd $(PF_ENGINE)/MUFFIN5/src/muffin5 or muffin5ebeta
% make

付 録B　システム拡張方法

B.1　Muffin 基本クラスライブラリ説明

Muffin 基本クラスライブラリのソースコードは、MUFFIN/src/common/ ディレクトリ以下にある。主なヘッダーファイルと機能概要を解説する。

DYNAMICS/DynamicsManager.h	ダイナミクスマネージャ
DYNAMICS/PartialRegionCondition.h	部分領域 (境界) 条件
FIELD/FDM_SuperField.h	FDM 用 場の基底クラス
FIELD/SuperField.h	FEM 用 場の基底クラス
IO/IOPath.h	コマンド引数の解析
IO/SuperStream.h	AVS 形式でのデータ出力
IO/PFIO/Imuffin3.h	MUFFIN3 の入出力 UDF オブジェクト
IO/PFIO/Imuffin3param.h	MUFFIN3 のコントロールパラメータ入力 UDF オブジェクト
IO/PFIO/Imuffin3result.h	MUFFIN3 の解析結果出力 UDF オブジェクト
IO/PFIO/PFIO.h	MUFFIN3 と UDF オブジェクト間のデータ IO
MESH/CreateMesh.h	さまざまな形状のメッシュ生成
MESH/ElementManager.h	メッシュ要素の集合体の操作
MESH/RegisterVertices.h	デローネ分割時の節点打ち込み関数
MESH/RightTriangulation3D.h	直方体系の 4 面体分割メッシュ
MESH/SuperElement.h	メッシュ要素の基底クラス
MESH/SuperMesh.h	メッシュの基底クラス
MESH/TriangleElement.h	三角形要素、四面体要素クラス
MESH/delaunaytriangulation.h	デローネ 2D 分割メッシュ
MESH/delaunaytriangulation3d.h	デローネ 3D 分割メッシュ
PARAM/IOParameterSet.h	パラメータセットクラス
SOLVER/FEMTetra.h	4 面体有限要素公式集 (補完関数の積分・微分) クラス
SOLVER/FEMTriangle.h	3 角形有限要素公式集 (補完関数の積分・微分) クラス
SOLVER/FiniteElement.h	有限要素公式集 (補完関数の積分・微分) 基底クラス
SOLVER/MATRIX/cg.h	共約勾配法クラス
UTIL/CheckMemorySize.h	使用メモリチェック関数
UTIL/MotherErr.h	エラー処理基底クラス
UTIL/ProgramTimer.h	計算時間チェック関数
UTIL/StringHelperFunc.h	文字列処理の UTILITY
UTIL/Sym2ndRankTensor.h	2 階対称テンソル型クラス
UTIL/Vector3d.h	ベクトル型クラス
UTIL/RND/rng.h	乱数基底クラス
UTIL/RND/twister.h	メルセンヌツイスター乱数
UTIL/DOCMAKER/muffin3docmaker.cpp	MUFFIN3 マニュアル自動生成ツール

```
UTIL/DOCMAKER/template_euc.tex      マニュアル自動生成テンプレート
SCRIPT/gxxversion.sh                コンパイラチェックスクリプト
-------------------------------------------------------------------------------
```

以上の中で、エンジン開発者が場のクラスの作成のために、把握しておいた方が良いのは、SuperField.h である。ここでは、SuperField.h のみを把握し、これを継承することで、新しい場のクラスの作成とエンジンへの追加が容易にできることを示す。

複雑な操作を望むエンジン開発者、特に任意のメッシュ要素の集合体を生成しそれを用いて解析をする場合には、更に、SuperMesh.h, (ElementManager.h,) SuperElement.h を把握した方が良い。

B.2 Muffin 基本クラスライブラリのコンパイルオプション

新しい自作のシミュレータを作成する場合には、今回リリースのシミュレータの main 関数など以下のソースを少しの修正で再利用するのが有効である。

```
-------------------------------------------------------------------------------
MUFFIN/src/Elastica/Makefile
MUFFIN/src/Elastica/MUFFIN.cpp        main 関数
MUFFIN/src/Elastica/Global_Variable.h main 関数のヘッダー
MUFFIN/src/Elastica/Fields/Fields.h   場のヘッダー
-------------------------------------------------------------------------------
```

次のような場を持つ、MyDynamics シミュレータを作成したい場合には、

- AField
- BField
- CField

以下のようにソースコードを構築する。Makefile, MUFFIN.cpp, Global_Variable.h, Fields.h は上記のものをコピー。

```
-------------------------------------------------------------------------------
MUFFIN3/src/MyDynamics/Makefile
MUFFIN3/src/MyDynamics/MUFFIN.cpp        main 関数
MUFFIN3/src/MyDynamics/Global_Variable.h main 関数のヘッダー
MUFFIN3/src/MyDynamics/Fields/Fields.h   場のヘッダー
MUFFIN3/src/MyDynamics/Fields/AField.h (and .cpp)
MUFFIN3/src/MyDynamics/Fields/BField.h (and .cpp)
MUFFIN3/src/MyDynamics/Fields/CField.h (and .cpp)
-------------------------------------------------------------------------------
```

Makefile には以下の行を追加する。

```
-------------------------------------------------------------------------------
###     For MUFFIN3 MyDynamics  ###
ifeq "$(THIS_TARGET)" "MyDynamics"
  # FEM シミュレータなら次の行を入れる
```

```
  #override MACRO += -D__UNSTRUCTURED_MESH__  -D__LAGRANGE_PICTURE__
  override MACRO += -D__UNSTRUCTURED_MESH__  -D__EULAR_PICTURE__
  SRC   = $(SRC001) $(SRC004) $(SRC005) $(SRC006) $(SRC007) $(SRC009) $(SRC010)
  # FDM シミュレータなら次の行を入れる
  #override MACRO += -D__STRUCTURED_MESH__  -D__EULAR_PICTURE__ -D__FDM__
  #SRC   = $(SRC001) $(SRC005) $(SRC006) $(SRC007) $(SRC010)
  SRCM  = MUFFIN2.cpp
  SRCS  = AField.cpp BField.cpp CField.cpp
  TARGET       = mydyn3
endif
```

MUFFIN.cpp の “Field Creation” 部分を以下の行に変える。

```
    //----------------------<<Field creation>>-----------------------
    // create fields
    AField af ;
    BField bf ;
    CField cf ;
    // Registration of fileds in dynamics manager
    a_Universe.RegisterField( &af );
    a_Universe.RegisterField( &bf );
    a_Universe.RegisterField( &cf );
```

Fields.h を以下のように変える。

```
#include ‘‘AField.h’’
#include ‘‘BField.h’’
#include ‘‘CField.h’’
```

以下に、Makefile で用いるコンパイルオプションとその効果について述べる。

B.2.1 使用する離散化手法とコンパイルオプション

各々のシミュレータで離散化の手法 (有限差分法 (FDM)、有限要素法 (FEM) など) が異なり、それに応じて Muffin 内部での場のデータの確保の方法が異なる。これら使用する離散化の手法 (FDM または FEM) は、次のようなシミュレータのコンパイルオプションで指定する。

- 有限差分法 (FDM) . . . __FDM__
- 有限要素法 (FEM) . . . 指定なし
  正確には、__FDM__ のコンパイルオプションが無い場合。

B.2.2 使用するメッシュタイプとコンパイルオプション

各々のシミュレータでシミュレーション描像や離散化の手法が異なり、それに応じて使用できるタイプのメッシュも限定される。これら使用するメッシュのタイプは、次のようなシミュレータのコンパイルオプションで指定する。

- 構造格子使用 (有限差分法シミュレータ) . . . __STRUCTURED_MESH__
  PhaseSeparation_FDM と Electrolyte_FDM で使用。使用できるメッシュタイプは “REGULAR”。
  ”RECTANGULAR”,”SPHERE1D”,”CYLINDER2D” は SUSI 用で MUFFIN ではサポートしていない。
- 非構造格子使用 (有限要素法シミュレータ) . . . __UNSTRUCTURED_MESH__
  PhaseSeparation_FEM, Electrolyte_FEM, Elastica, GelDyna で使用。使用できるメッシュタイプは、
  “UNSTRUCTURED_RECT”,
  ”UNSTRUCTURED_INPUT”。更に、以下のメッシングアルゴリズムをコンパイルオプションで取り入れ可能。
  - 2 次元デローネ分割アルゴリズム使用 (2D アダプティブ有限要素法) . . . __DELAUNAY2D_MESH__
    MUFFIN 共通ライブラリでサポート済み。更に、使用できるメッシュタイプは
    ”UNSTRUCTURED_SPHERE”。
  - 3 次元デローネ分割アルゴリズム使用 (3D アダプティブ有限要素法) . . . __DELAUNAY3D_MESH__
    今回リリースでは無い。更に、使用できるメッシュタイプは、”UNSTRUCTURED_SPHERE”,
    “UNSTRUCTURED_CYLINDER”。

これらを指定すると、必要なタイプのメッシュ生成プログラムだけが一緒にコンパイルされる。指定が無いと、すべてのタイプのメッシュ生成プログラムが一緒にコンパイルリンクされるため、モジュールサイズが非常に (不要に) 大きくなり、コンパイル時間も長くなる。

また、非構造格子 __UNSTRUCTURED_MESH__ マクロがある (正確には 構造格子 __STRUCTURED_MESH__ マクロがない) 場合にのみ、メッシュ構造データが出力される。

B.2.3 シミュレーション描像とコンパイルオプション

メッシュ構造データは非構造格子でのみ出力される。これらのメッシュ構造データのコモンデータ、レコードデータへの出力の振り分けは、シミュレーション描像に依存しており、コンパイルオプションでコントロールされる。

PFIO.cpp のソースのコンパイルのコンパイルオプションで、

- オイラー描像 . . . __EULER_PICTURE__
  すべてコモンデータ。
- ラグランジュ描像 . . . __LAGRANGE_PICTURE__
  節点座標データのみレコードデータ。
- アダプティブメッシュ . . . __ADAPTIVE_MESH__
  すべてレコードデータ。

とすると、メッシュデータのコモンおよびレコードへの出力がコントロールされる。これらのコンパイルオプション無しにコンパイルすると、すべてのメッシュデータがコモンデータ、レコードデータの両方に出力され出力データサイズが大きくなる。

メッシュの近傍要素情報の出力の有無は、PFIO.cpp のソースのコンパイルオプションの
__NEIGHBOR_DATA_IO__ でコントロールされる。このコンパイルオプションがある場合に、出力 UDF に近傍データが出力される。

B.3 機能拡張チュートリアル (入門編) - 新しい場のクラスの作成法 -

新しい場のクラスは、SuperField.h をインクルードし、クラス PhysicalField を継承することで定義する。

以下に、AField.h を例に場のクラスの書き方と用意しているインターフェースの解説を行う。

以下は入門編で、SuperField が提供するパラメータの取得、他の場のデータの取得、部分領域 (境界) 境界条件の条件に関する基本的な関数を解説する。

```
-------------------------------------------------------------------------------
//-----------------------------------------------------------
// AField.h                                by T.Yamaue
//-----------------------------------------------------------
#ifndef _AFIELD_INCLUDED_
#define _AFIELD_INCLUDED_

#include "SuperField.h"

class AField : public PhysicalField<double> // 場の型は double
{
public:
  // デフォルトコンストラクタ
  AField(void):PhysicalField<double>("AField") // 場の名前
  { SetFunctionName(); }
  virtual int Evolve00(void) // (1Step) 時間発展コマンド (SOLVE:NONSENSE_SUM)
  {
    //他の Field は名前で取得できる
    SuperField& b_field = Field("BField"); // 型 double とする
    SuperField& c_field = Field("CField"); // 型 Vector とする

    //パラメータは (MeshParameter, PhysicalParameter などによらず) 名前で取得できる。
    double param_double = Parameter().value("Double_Parameter");
    string param_string = Parameter().stering_value("String_Parameter");
    bool param_bool = Parameter().bool_value("Bool_Parameter");

    // この場が乗っているメッシュ要素のリストは
    // vector<SuperElement*> SuperField::ObjList(void) で取得
    // すべての場の値を パラメータ Double_Parameter と、
    // 場 BField (型 double) の同じメッシュ要素での値と、
    // 場 CField (型 Vector) の X 座標の値の和に設定するとき。
    for (vector<SuperElement*>::const_iterator i = ObjList().begin() ;
      i != ObjList().end() ; i++) {
        value(*i) // 自分の値は型付きで返る
          = param_double + b_field.value(*i)
            // 他の場は第 2 引数が無ければ double で返ってくる。
            + (c_field3.value(*i, OVector)).x  ;
            // 他の double 以外の型の場は、第 2 引数に
            // OVector, OSym2ndRankTensor などと書くと、その型で返ってくる。
```

```
    }
    return 1 ;
  }
  virtual int Evolve01(void) // (1Step) 時間発展コマンド (SOLVE:NONSENSE_EMPTY)
  { return 1 ;}
  virtual int Initialize00(void)  // 初期化コマンド (INITIAL:NONSENSE_EMPTY)
  { return 1 ;}

protected:
  void SetFunctionName(void)
  {
    // コマンドの名前を登録
    SetSolverName("SOLVE:NONSENSE_SUM") ;
    SetSolverName("SOLVE:NONSENSE_EMPTY") ;  // 00,01,...99 まで上から順番に
    SetInitializingFuncName("INITIAL:NONSENSE_EMPTY") ;
  }
};
#endif
-------------------------------------------------------------------------------
```

必要な場について、このような場のクラスを構築し、シミュレータを make する。

続いて、入力 UDF のダイナミクスマネージャ部で、場のクラスの登録、プロシージャの登録を行なう。詳しくは入力 UDF 解説を参照。

B.4 システム拡張のための有限要素法公式集

線形補完の有限要素法を行う際の、1 次補完関数の積分・微分式は

```
-------------------------------------------------------------------------
SOLVER/FEMTetra.h                4 面体有限要素公式集 (補完関数) クラス
SOLVER/FEMTriangle.h             3 角形有限要素公式集 (補完関数) クラス
SOLVER/FiniteElement.h           有限要素公式集 (補完関数) 基底クラス
-------------------------------------------------------------------------
```

のクラスでライブラリ化されており、独力で計算する必要はない。ここでは、これらのライブラリの機能である有限要素公式集を載せておく。新しいソルバを追加するために、有限要素法で問題を離散化する際などに便利であろう。

以下とくに断りのない場合 2 次元、3 次元共通に以下のような表示を用いる。

- $\boldsymbol{r}$: 空間座標
- $\boldsymbol{v}, \boldsymbol{u}$: 速度
- $i, j, k, ...$: 空間次元を表すインデックス
- $I, J, K, ..$: 一つの要素内の節点を表すインデックス。
- ∇, ∇_i : 空間微分 (ナブラ) およびその i 座標成分。

- $L_I(\boldsymbol{r})$：要素内一次補間関数 (節点 I で値が 1)。要素内面積または体積座標も意味する。
- $\Phi_I(\boldsymbol{r}), \Psi_I(\boldsymbol{r})$：一次補間関数を含む一般の要素内補間関数。
- $\int dV$：3 次元体積積分
- $\int dA$：2 次元面積積分
- $\int d\boldsymbol{r}$：(全) 空間積分。
- $\int dS$：表面積分。
- $\boldsymbol{n}, n_i$：表面法線ベクトルとその成分。
- A_e：要素の面積 (2 次元)
- V_e：要素の体積 (3 次元)
- $\int_e dV, \int_e d\boldsymbol{r}, \int_e dS$：要素内空間積分。要素表面積分。
- Ω_e：要素の面積 (2 次元) または体積 (3 次元)。要素そのものを示す場合もある ($\boldsymbol{r} \in \Omega_e$ など)。
- p_e：要素を構成する節点の数

2 回現れるテンソルインデックスに関する縮約を行うかどうかはその都度明記する。

B.4.1 一次補間関数の積分

一次補間関数に関しては以下の式で積分を評価することができる。

$$\int_e L_1^m L_2^n dx = \frac{m!n!}{(m+n+1)!} D_e \quad (1\text{ 次元要素}) \tag{B.1}$$

$$\int_e L_1^l L_2^m L_3^n dA = \frac{l!m!n!}{(l+m+n+2)!} 2A_e \quad (2\text{ 次元三角要素}) \tag{B.2}$$

$$\int_e L_1^k L_2^l L_3^m L_4^n dV = \frac{k!l!m!n!}{(k+l+m+n+3)!} 6V_e \quad (3\text{ 次元四面体要素}) \tag{B.3}$$

いくつかの特別なケースについて；

$$\int_e L_I d\Gamma = \Omega_e / p_e \quad (\text{任意の次元}) \tag{B.4}$$

$$\int_e L_I L_J dx = \frac{1+\delta_{IJ}}{6} D_e \quad (1\text{ 次元要素}) \tag{B.5}$$

$$\int_e L_I L_J dA = \frac{1+\delta_{IJ}}{12} A_e \quad (2\text{ 次元三角要素}) \tag{B.6}$$

$$\int_e L_I L_J dV = \frac{1+\delta_{IJ}}{20} V_e \quad (3\text{ 次元四面体要素}) \tag{B.7}$$

B.4.2 一次補間関数の微分

2 次元三角形要素、3 次元四面体要素に関して節点 I に対向する辺の長さまたは面の面積を B_I とするとき、要素の面積または体積は

$$\Omega_e = \frac{h_I B_I}{g} \tag{B.8}$$

ここで h_I は節点 I の対向する辺または面からの「高さ」で g は三角形要素で 2、四面体要素で 3。この場合

$$|\nabla L_I| = \frac{1}{h_I} \tag{B.9}$$

である。

- 2 次元三角形一次要素

  節点 I に向かい合う節点を $I+1$,$I+2$ (反時計周り) とすると。

$$\nabla L_I(\boldsymbol{r}) = \frac{1}{h_I}\frac{1}{B_I}\begin{bmatrix} y_{I+2}-y_{I+1} \\ x_{I+1}-x_{I+2} \end{bmatrix} = \frac{1}{2A_e}\begin{bmatrix} y_{I+2}-y_{I+1} \\ x_{I+1}-x_{I+2} \end{bmatrix} \tag{B.10}$$

  接点の座標を $\boldsymbol{r}_I, \boldsymbol{r}_{I+1}, \boldsymbol{r}_{I+2}$ として以下のようにも表わせる。

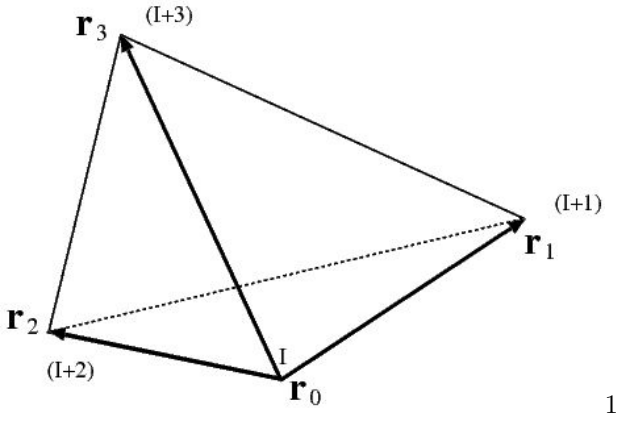
$$\nabla L_I(\boldsymbol{r}) = \frac{1}{2A_e}\boldsymbol{R}(\boldsymbol{r}_{I+2}-\boldsymbol{r}_{I+1}) \tag{B.11}$$

$$\nabla L_{I+1}(\boldsymbol{r}) = \frac{1}{2A_e}\boldsymbol{R}(\boldsymbol{r}_{I}-\boldsymbol{r}_{I+2}) \tag{B.12}$$

$$\nabla L_{I+2}(\boldsymbol{r}) = \frac{1}{2A_e}\boldsymbol{R}(\boldsymbol{r}_{I+1}-\boldsymbol{r}_{I}) \tag{B.13}$$

$$\boldsymbol{R} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{B.14}$$

- 3 次元四面体一次要素



1.2

図 B.1: 3 次元四面体要素での節点配置

節点 $\boldsymbol{r}_I = \boldsymbol{r}_0$ に向かい合う節点の座標ベクトルを図 B.1 に示すように $\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3$ とすると、

$$\begin{aligned}\nabla L_I(\boldsymbol{r}) &= \pm\frac{1}{h_I}\frac{(\boldsymbol{r}_3-\boldsymbol{r}_1)\times(\boldsymbol{r}_2-\boldsymbol{r}_1)}{|(\boldsymbol{r}_3-\boldsymbol{r}_1)\times(\boldsymbol{r}_2-\boldsymbol{r}_1)|}\\ &= \pm\frac{1}{h_I}\frac{(\boldsymbol{r}_3-\boldsymbol{r}_1)\times(\boldsymbol{r}_2-\boldsymbol{r}_1)}{2B_I}\\ &= \pm\frac{1}{6V_e}(\boldsymbol{r}_3-\boldsymbol{r}_1)\times(\boldsymbol{r}_2-\boldsymbol{r}_1)\end{aligned}$$

ただし ± の符号は

$$(\boldsymbol{r}_0-\boldsymbol{r}_1)\cdot(\boldsymbol{r}_3-\boldsymbol{r}_1)\times(\boldsymbol{r}_2-\boldsymbol{r}_1)$$

の符号に合わせて付ける。

まとめると；

$$\nabla L_I(\boldsymbol{r}) = \frac{1}{6V_e}(\boldsymbol{r}_3-\boldsymbol{r}_1)\times(\boldsymbol{r}_2-\boldsymbol{r}_1) \tag{B.15}$$

$$\nabla L_{I+1} = \frac{1}{6V_e}(\boldsymbol{r}_2-\boldsymbol{r}_0)\times(\boldsymbol{r}_3-\boldsymbol{r}_0) \tag{B.16}$$

$$\nabla L_{I+2} = \frac{1}{6V_e}(\boldsymbol{r}_3-\boldsymbol{r}_0)\times(\boldsymbol{r}_1-\boldsymbol{r}_0) \tag{B.17}$$

$$\nabla L_{I+3} = \frac{1}{6V_e}(\boldsymbol{r}_1-\boldsymbol{r}_0)\times(\boldsymbol{r}_2-\boldsymbol{r}_0) \tag{B.18}$$

B.4.3 一次補間関数の要素境界での面積分

境界条件処理には、

$$\lambda_{IJ} = \int_{S_J} dS L_I(\boldsymbol{r}) \tag{B.19}$$

の形の面積分が必要となる。ここで S_J は要素上で節点 J に対向する辺 (三角要素) または面 (四面体要素) を示す。

明らかに

$$\lambda_{II} = \int_{S_I} dS L_I(\boldsymbol{r}) = 0 \tag{B.20}$$

である。

- 2 次元三角形要素節点 $J \neq I$ に対して辺 S_J 上では $L_I(\xi) = (1-\xi/B_J)$ である。ここで ξ は節点 I から測った辺 S_J 上の点での距離。故に；

$$\lambda_{IJ} = \int_0^{B_J}(1-\xi/B_J)d\xi = B_J/2 \tag{B.21}$$

- 3 次元四面体要素節点 $J \neq I$ に対して面 S_J 上では $L_I(\xi) = (1-\xi/\gamma_J)$ である。ここで ξ は面 S_J 上で節点 I から引いた垂線上での距離で $0 < \xi < \gamma_J$。また ξ での積分に対する面積要素は β_J を S_J 上で I に対向する辺の長さとして $\beta_J\xi d\xi$。故に；

$$\lambda_{IJ} = \int_0^{\gamma_J}(1-\xi/\gamma_J)\beta_J\xi d\xi = (1/6)\beta_J\gamma_J = B_J/3 \tag{B.22}$$

一般的に考えればこれらは一つ次元が低い空間での一次要素の積分と考えれば良いことが分かる。それゆえより一般的な形では；

- 2 次元三角要素

$$\lambda_{IJK}^{mn} = \int_{S_K} dS L_I^m L_J^n = (1 - \delta_{IK}\delta_{JK}) \frac{m!n!}{(m+n+1)!} B_K \tag{B.23}$$

- 3 次元四面体要素

$$\lambda_{IJKM}^{lmn} = \int_{S_M} dS L_I^l L_J^m L_K^n = (1 - \delta_{IM}\delta_{JM}\delta_{KM}) \frac{l!m!n!}{(l+m+n+2)!} 2B_M \tag{B.24}$$