



ユーザーマニュアル

リリース **5.3**

Yves Renard, Julien Pommier, Konstantinos Poullos 著
Tetsuo Koyama 訳

2018 年 09 月 26 日

目次

第 1 章	イントロダクション	1
第 2 章	インストール方法	3
第 3 章	線形代数プロセス	5
第 4 章	<i>GetFEM++</i> の MPI 並列化	7
4.1	<i>GetFEM++</i> MPI の並列化の進捗状況	8
第 5 章	キャッチエラー	11
第 6 章	メッシュ生成	13
6.1	メッシュに要素を追加する	13
6.2	メッシュから要素を削除する	16
6.3	シンプルな構造化メッシュ	16
6.4	メッシュ領域	17
6.5	<code>getfem::mesh</code> オブジェクトのメソッド	17
6.6	<code>dal::bit_vector</code> を使います	20
6.7	面番号	21
6.8	メッシュの保存とロード	21
第 7 章	メッシュ上に有限要素法を構築します	25
7.1	第 1 レベル: 各要素の <code>fem</code> の操作	26
7.2	例	27
7.3	第 2 レベル: オプションである「ベクトル化/テンソル化」	28
7.4	第 3 レベル: オプションである線形 (または縮小) 変換	29
7.5	汎用的な <code>mesh_fem</code> の取得	30
7.6	<code>partial_mesh_fem</code> オブジェクト	31
第 8 章	積分方法の選択	33
8.1	<code>mesh_im</code> オブジェクトのメソッド	35
第 9 章	メッシュの改善	37

第 10 章 任意の項を計算する - 高水準の汎用的な構築手順 - 弱形式言語	39
10.1 高水準と低水準の汎用構築間の実行時間の差	39
10.2 弱形式言語構文の概要	40
10.3 いくつかの基本的な例	42
10.4 導出順序と記号による微分	43
10.5 構築のための C++ 呼び出し	44
10.6 C++ 構築の例	45
10.7 構築言語のスクリプト言語の呼び出し	48
10.8 テンソル	49
10.9 変数	49
10.10 定数またはデータ	49
10.11 試行関数	50
10.12 勾配	50
10.13 Hessian	50
10.14 定義済みのスカラー関数	50
10.15 ユーザー定義のスカラー関数	51
10.16 定義されたスカラー関数の導関数	52
10.17 バイナリ演算	52
10.18 単項演算子	53
10.19 括弧	53
10.20 陽なベクトル	53
10.21 陽な行列	54
10.22 陽なテンソル	54
10.23 テンソル成分へのアクセス	54
10.24 定数式	54
10.25 現在の位置にリンクされた特殊な表現	55
10.26 プリントコマンド	55
10.27 テンソルを変形する	55
10.28 Trace, Deviator, Sym と Skew 演算子	56
10.29 非線形演算子	56
10.30 マクロ定義	57
10.31 陽な微分	58
10.32 陽な勾配	59
10.33 補間変換	59
10.34 要素外挿変換	61
10.35 要素間エッジ/面間の不連続性の評価します。	62
10.36 ダブル領域の積分または項 (畳み込み - カーネル - 交換積分)	63
10.37 初等変換	64

10.38 Xfem 不連続性評価 (mesh_fem_level_set を使用)	65
10.39 構築中の getfem::im_data オブジェクトへのサブ式の格納	66
第 11 章 任意の項を計算する - 低レベルの汎用的な構築手順	67
11.1 comp コマンドの中で利用可能な操作	70
11.2 他の操作	70
第 12 章 いくつかの標準構築手順 (低レベル汎用構築)	71
12.1 Laplacian (Poisson) 問題	71
12.2 線形弾性問題	73
12.3 混合有限要素法によるストークス問題	74
12.4 質量行列の構築	74
第 13 章 任意の量の補間	75
13.1 基本補間	75
13.2 高レベル弱形式言語に基づく補間	76
第 14 章 GetFEM++ に新しい有限要素法を組み込む	79
第 15 章 GetFEM++ に新しい近似積分法を組み込む	81
第 16 章 レベル集合法、拡張有限要素法 (Xfem)、仮想領域有限要素法、切断有限要素法	83
16.1 レベル集合法の表現	84
16.2 レベル集合法によるメッシュ切断	84
16.3 統合された積分方法	85
16.4 切断有限要素法	86
16.5 いくつかのレベル集合法にわたる不連続なフィールド	86
16.6 拡張有限要素法 (Xfem)	87
16.7 ポスト処理	88
第 17 章 非適合メッシュ上の有限要素法の補間	89
17.1 異なるメッシュの混合メソッド	90
17.2 モルタル法	90
第 18 章 L^2 と H^1 ノルムの計算	91
第 19 章 導関数の計算	93
第 20 章 解の出力と表示	95
20.1 Matlab インターフェース用の mesh と mesh_fem オブジェクトの保存	95
20.2 メッシュスライスの生成	96
20.3 mesh か mesh_fem またはスライスを VTK に出力します	98

20.4	<i>mesh</i> か <i>mesh_fem</i> またはスライスを OpenDX に出力する	99
第 21 章	純対流法	101
第 22 章	モデル記述と基本モデルブリック	103
22.1	<i>model</i> オブジェクト	103
22.2	<i>brick</i> オブジェクト	106
22.3	新しい項を作る方法	107
22.4	モデルに項を追加する方法	111
22.5	汎用的な構築ブリック	113
22.6	汎用的な楕円ブリック	114
22.7	Dirichlet 条件ブリック	115
22.8	一般化 Dirichlet 状態ブリック	117
22.9	点列制約ブリック	118
22.10	ソース項ブリック (および Neumann 条件)	119
22.11	あらかじめ定義されたソルバー	119
22.12	Poisson 問題の完全な例	120
22.13	Dirichlet および接触境界条件に対する Nitsche 法	122
22.14	拘束ブリック	126
22.15	他の“陽な”要素	126
22.16	Helmholtz 要素	127
22.17	Fourier-Robin 要素	127
22.18	等方性線形弾性ブロック	128
22.19	線形非圧縮性 (またはほぼ非圧縮性) ブリック	129
22.20	質量ブリック	131
22.21	Bilaplacian と Kirchhoff-Love プレートブリック	131
22.22	Mindlin-Reissner プレートモデル	132
22.23	過渡問題の積分のためのモデルツール	135
22.24	摩擦ブリックとの微小すべり接触	142
22.25	摩擦ブリックとの有限すべり/有限変形接触	156
第 23 章	数値連続法と分岐	165
23.1	数値連続法	165
23.2	限界点の検出	168
23.3	数値分岐	168
23.4	モデルの解曲線の近似	171
第 24 章	有限歪弾性ブリック	173
24.1	有限歪み弾性に関するいくつかの復習	173
24.2	モデルに非線形弾力性ブリックを追加する	178

24.3	モデルに大ひずみの非圧縮性ブリックを追加する	179
24.4	高レベル汎用構築バージョン	180
第 25 章	微小ひずみの可塑性	183
25.1	理論的背景	183
25.2	流れ則の積分	185
25.3	いくつかの古典的な法則	190
25.4	弾塑性ブリック	195
第 26 章	剛体運動が大きい物体の ALE サポート	201
26.1	物体を回転させるための ALE 項	201
26.2	一様変形物体の一部の ALE 項	205
第 27 章	付録 A. 有限要素法リスト	207
27.1	シンプレックスの古典的 Lagrange 要素 P_K	209
27.2	他のジオメトリ上の古典的な Lagrange 要素	212
27.3	ハイアラーキ基底の要素	216
27.4	古典的なベクトル要素	220
27.5	次元 1 の特定の要素	222
27.6	次元 2 の特定の要素	223
27.7	次元 3 の特定の要素	234
第 28 章	付録 B. 立体求積法のリスト	241
28.1	完全な積分法	241
28.2	Newton Cotes 積分法	242
28.3	次元 1 の Gauss 積分法	242
28.4	次元 2 の Gauss 積分法	242
28.5	次元 3 の Gauss 積分法	246
28.6	積分法の直積	248
28.7	具体的な積分法	248
28.8	コンポジット積分法	248
第 29 章	References	251
	参考文献	253
	索引	257

第 1 章

イントロダクション

GetFEM++ プロジェクトは、一般的で効率的な有限要素法の基本計算のための C++ ライブラリです。目的は最大級の手法と要素、そして任意の次元（つまり、2次元と3次元の問題に限らず）について任意の基本行列の計算（混合型有限要素法の場合でも）を可能にするライブラリを与えることです。

このライブラリでは、積分法（厳密または近似）、幾何変換（線形または非線形）、および任意の次元の有限要素法が完全に分離されています。それにより、基本的な計算が技術的に困難なより積分された有限要素法コーディングを楽にすることができます。

利用可能な有限要素法は、任意の次数と次元の単純な P_k 、平行 6 面体の Q_k 、気泡関数を持つ P_1 、 P_2 、Hermite 要素、階層ベースの要素（例えばマルチグリッド法の場合）、不連続な P_k または Q_k 、XFem、Argyris、HCT、Raviart-Thomas などです。

新しい有限要素法を追加するのは簡単です。参照要素に関する記述が必要になります（ほとんどの場合、これは基底関数の説明であり、それ以上は必要ありません）。拡張機能として Hermite 要素、微分的多項式、非多項式およびベクトル要素、XFem が提供されています。

ライブラリには、古典的な偏微分方程式の組み立て手順、補間法、ノルムの計算、メッシュ操作、境界条件、メッシュからのスライスの抽出などのポスト処理ツールなど、有限要素法用の通常のツールも含まれています。

GetFEM++ はとても汎用的な有限要素コードを作成するために使用できます。有限要素、積分法、メッシュの次元は、非常に簡単に変更できるいくつかのパラメータであり、したがって実験の幅が広がります。多数の例が、配布物の `tests` ディレクトリにあります。

GetFEM++ は（非常に）実験的なメッシュ化手順しか持たず（そして通常メッシュを生成します）、メッシュはインポートするのが一般的です。*GetFEM++* で現在知られているインポート形式は、*GiD*、*Gmsh* および *emc2* メッシュファイルです。ただし、メッシュを指定すると、自動的に修正することができます。

Copyright © 2004-2018 *GetFEM++* project.

GetFEM++ のウェブサイトのテキストとドキュメントは、GNU Free Documentation License の条件の下で変

更および再利用することができます。(訳者注: 本ドキュメントの翻訳は以下のプロジェクトで行われています。
<https://www.transifex.com/getfem-doc/getfem-53-1>)

このライブラリはフリーソフトウェアです。あなたはこれを、フリーソフトウェア財団によって発行された GNU 劣等一般公衆利用許諾契約書 (バージョン 3 のライセンス、または (あなたのオプションで) いずれかのそれ以降のバージョンを、GCC ランタイムライブラリ例外とともに、バージョン 3.1 または (あなたのオプションで) いずれかのそれ以降のバージョンに置き換えてください。) の定める条件の下で再頒布または改変することができます。このライブラリは有用であることを願って頒布されますが、全くの無保証です。商業可能性の保証や特定の目的への適合性は、言外に示されたものも含め全く存在しません。詳しくは GNU 劣等一般公衆利用許諾契約書をご覧ください。あなたはこのライブラリと共に、GNU 劣等一般公衆利用許諾契約書の複製物を一部受け取ったはずですが、もし受け取っていない場合は、フリーソフトウェア財団まで請求してください (宛先は the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA)。

第 2 章

インストール方法

標準的な *GNU* ツールを使用しているので、*GetFEM++* ライブラリのインストールは幾分標準的です。異なるプラットフォーム上のインストールの詳細については、[ダウンロードとインストール ページ](#)を参照してください。

第 3 章

線形代数プロセス

GetFEM++ で使用される線形代数ライブラリは *Gmm++* で、今は別のライブラリとして独立しています。[GMM++ ユーザドキュメント](#) を参照してください。

GetFEM++ 内にはバージョン 3.0 の *SuperLU* ([SuperLU のウェブサイト](#) を確認してください) が含まれているので (リリース 1.7 以降)、直接的なスパースソルバーがそのまま利用可能です。ただし、`./configure` ファイルのオプションでは、インストールされているバージョンを使用するために *SuperLU* のインクルードバージョンを無効にすることができます。

MUMPS に対する小さなインターフェースも提供されています ([MUMPS のウェブサイト 1](#) もしくは [MUMPS のウェブサイト 2](#) を確認してください)。ファイル `gmm/gmm_MUMPS_interface.h` を参照してください。*MUMPS* を使用するには、`configure` シェルにいくつかのオプションを指定する必要があります

```
--with-mumps-include-dir=" -I /path/to/MUMPS/include "
--with-mumps=" F90 libraries and libs of MUMPS to be linked "
```

あるいは、オプション `--enable-mumps` を指定することで、インストールされている *MUMPS* ライブラリを検索します。あなたのシステム (特に *Debian* と *Ubuntu*) にシーケンシャル版と並列版の両方がインストールされている場合、デフォルトのバージョンは並列版になります。シーケンシャル版を選択するには、オプション `--with-mumps="-lsmumps_seq -ldmumps_seq -lcmumps_seq -lzmumps_seq"` を追加する必要があります。

例えば、*MUMPS* のシーケンシャル版を倍精度と倍精度複素数で使いたい場合

```
--with-mumps-include-dir=" -I /path/to/MUMPS/include "
--with-mumps=" ...F90libs... -L /path/to/MUMPS/lib -ldmumps -lzmumps -lpord
               -L /path/to/MUMPS/libseq -lmpiseq "
```

ここで、`...F90libs...` は、*MUMPS* をコンパイルするために使用される Fortran コンパイラのライブラリです (この部分は使用されている Fortran 90 コンパイラに大きく依存しますが、`./configure` スクリプトではマシン上のデフォルトの Fortran 90 コンパイラに関連するオプションを捜し出して指定する必要があります)。

例えば、ifort コンパイラでは、`-L/opt/icc8.0/lib -lifport -lifcoremt -limf -lm -lcxa -lunwind -lpthread` です。)

第 4 章

GetFEM++ の MPI 並列化

当然ながら、各問題は、良好な負荷分散を行うために、それぞれ固有な事情に合わせて異なる並列化を必要となります。このライブラリではメッシュ領域を使用して自身で並列化を構行い、行列の組み立て手順を並列化することができます。

併せて、ブリック系は MPI(プロセス間の通信)、METIS (メッシュの分割) および MUMPS (並列化された疎行列直接解法) に基づく汎用的な並列化を提供しています。この機能はコンパイラオプション `-D GETFEM_PARA_LEVEL = 2` で利用でき、ライブラリ自体は `configure` スクリプトの `--enable-paralevel=2` オプションでコンパイルする必要があります。GetFEM++ の初期の MPI 並列化は、CALMIP 大学の Nicolas Renon と Toulouse の助けを借りて設計されています。

`configure` スクリプトを `--enable-paralevel=2` オプションで実行すると、MPI、METIS、および並列 MUMPS ライブラリが検索されます。Python インタフェースが構築されている場合は、MPI4PY ライブラリも検索します。その場合、python インタフェースを使用して `getfem` の並列版を実行することができます (他のインタフェースは現時点では並列化されていません)。`interface/test/python` ディレクトリの `demo_parallel_laplacian.py` を参照してください。

`-D GETFEM_PARA_LEVEL = 2` オプションを使用すると、使用される各メッシュは (METIS を使用して) プロセッサの数に対応するいくつかの領域に暗黙的に分割され、構築手順が並列化されます。つまり、`model_state` 変数にアセンブルされた接線行列と制約行列が分散されているということです。(今のところ) ベクトルを分散することはできません。`model_state` 変数の右手側のベクトルが各プロセッサに伝達されるようになっています (それぞれの計算結果の積分は構築の最後で行われ、各プロセッサは完全なベクトルを持ちます)。ブリックにより格納された行列はすべて分散されているということに注意してください。

C++ の並列プログラムの雛形は `tests/elastostatic.cc` です。並列で計算するためには次のように実行します

```
mpirun -n 4 elastostatic elastostatic.param
```

Python インタフェースのプログラムの場合、呼び出しは

```
mpirun -n 4 python demo_parallel_laplacian.py
```

`make install` を実行しない場合は、最初にシェル変数 `PYTHONPATH` を `python-getfem` ライブラリに設定することを忘れないでください

```
export PYTHONPATH=my_getfem_directory/interface/src/python
```

4.1 *GetFEM++* MPI の並列化の進捗状況

`getfem` の並列化は引き続き「進行中の作業」になります。一定数のプロセスが依然としてシーケンシャルです。ご存知のとおり、プログラムの並列化が正確かどうかを確認する良いテストは、計算結果がプロセスの数とは無関係であることを検証することです。

- 構築手順

構築手順の大部分は (`getfem/getfem_assembling.h` を参照) 構築が計算される領域に対応するパラメータを持っています。それらは並列化されていませんが、ジョブを分配するために各プロセスで異なる領域を呼ぶことが望ましいです。ファイル `getfem/getfem_config.h` には分散疎行列の処理結果を集約するための `MPI_SUM_SPARSE_MATRIX` というプロシージャが含まれています。

以下の構築プロシージャは、`-D GETFEM_PARA_LEVEL=2` オプションを使用して暗黙的に並列化されます。

- ノルムの計算 (`getfem/getfem_assembling.h` 内の `asm_L2_norm`, `asm_H1_norm`, `asm_H2_norm`)、
- `asm_mean_value` (`getfem/getfem_assembling.h` を参照)、
- `error_estimate` (`getfem/getfem_error_estimate.h` を参照)。

つまり、これらの関数を各プロセッサで呼び出す必要があります。

- `Mesh_fem` オブジェクト

`getfem :: mesh_fem` オブジェクトの自由度ナンバリングは連続のまま、各プロセスで実行され、並列化が行われます。これは、非常に大きなメッシュまたは縮閉メッシュの並列化の効率に影響する可能性があります。

- `Model` オブジェクトとブリック

モデルシステムは全体的に並列化されています。これは主に、標準ブリックの構築プロシージャがメッシュの `METIS` パーティションを使用して構築を分配しているためです。正接/剛性行列は分散したままであり、標準解法は `MUMPS` (分散行列を受け入れる) の並列版が呼び出されます。

現時点では、`model` オブジェクトの `actualize_sizes ()` プロシージャはシーケンシャルなままで、各プロセスで実行されます。並列化はこれからです。

いくつかの特異性:

- 明示的な質量行列: 与えられた行列は分配されているものとみなされます。そうでない場合は、マスタープロセスにのみ追加してください (その他の場合は、プロセスの数で寄与分が掛けられます)。
- 明示的な右辺 (rhs) ブリック: 与えられたベクトルは分布しているとはみなされません。マスタープロセス上の指定されたベクトルのみが考慮されます。
- 拘束ブリック: 与えられた行列と右辺 (rhs) は分配されているとはみなされません。マスタープロセス上の与えられた行列とベクトルのみが考慮されます。
- 接触ブリックに関しては、一体型接触ブリックのみが完全に並列化されています。節点接触ブリックは並行で動作しますが、すべての計算はマスタープロセスで実行されています。

第 5 章

キャッチエラー

GetFEM++ で使用されるエラーは `gmm / gmm_except.h` ファイルで定義されます。エラーをより簡単にするために、すべてのエラーは S.T.L のファイル `stdexcept` に定義された `std::logic_error` 型から得られます。

標準的な手順である `GMM_STANDARD_CATCH_ERROR` は `gmm/gmm_except.h` で定義されています。このプロセスは、すべてのエラーをキャッチし、エラーが発生したときにエラー・メッセージを出力します。次のようにプログラムのメインプロセスで使用できます:

```
int main(void) {
    try {
        ... main program ...
    } GMM_STANDARD_CATCH_ERROR;
}
```


第 6 章

メッシュ生成

前置きとして、*GetFEM++* 用語に関する簡単な説明をします。

GetFEM++ はファイル `getfem/bgeot_mesh_structure.h` と `getfem/getfem_mesh.h` で定義されるメッシュを格納する固有の構造を持っています。

このオブジェクトは、異なる次元の要素を混合しても、任意の次元に任意の要素を格納できます。

複雑なジオメトリをメッシュ化するには、*GetFEM++* には（非常に）実験的なメッシング手順しかありません。しかし、どんなフォーマットからでもメッシュを簡単に読み込むことができます（いくつかのプロシージャは、`getfem/getfem_import.h` にあり、いくつかのパブリック領域メッシュジェネレータからメッシュをロードします）。

構造体 `getfem::mesh` は、境界や要素の集合など、メッシュの領域に関する記述を含むこともできます。これは、凸面と凸面のコンテナ `getfem::mesh_region` を介して処理されます。

6.1 メッシュに要素を追加する

変数 `mymesh` が次のように定義されているとします。

```
getfem::mesh mymesh;
```

このメッシュに新しい要素を挿入するには、ポイントのリストから追加する方法と既存のポイントのインデックスのリストから追加する 2 つの方法があります。

メッシュ上に新しい点を入力するには、次のメソッドを使用してください。

```
i = mymesh.add_point(pt);
```

ここで、`pt` は `bgeot::base_node` 型です。インデックス `i` は、メッシュ上のこのポイントのインデックスです。すでにメッシュ内にポイントが存在する場合、新しいポイントは挿入されず、既存のポイントのインデッ

クスが返されます。メッシュには、その点の次元である主次元があります。同じメッシュ内に異なる次元の点を持つことはできません。

新しい要素をメッシュに追加する最も基本的な関数は次の通りです。

```
j = mymesh.add_convex(pgt, it);
```

これは、`bgeot::pgeometric_trans` 型の `pgt` (基本的には `bg_gt` 型のインスタンスへのポインタ) を持つテンプレート関数であり、既に存在する点のインデックスのリスト上の反復子です。たとえば、3次元メッシュに新しい3角形を追加する必要がある場合、最初に3つの点のインデックスを持つ配列を定義する必要があります

```
std::vector<bgeot::size_type> ind(3);
ind[0] = mymesh.add_point(bgeot::base_node(0.0, 0.0, 0.0));
ind[1] = mymesh.add_point(bgeot::base_node(0.0, 1.0, 0.0));
ind[2] = mymesh.add_point(bgeot::base_node(0.0, 0.0, 1.0));
```

要素の追加は、次のように行います。

```
mymesh.add_convex(bgeot::simplex_geotrans(2,1), ind.begin());
```

ここで `bgeot::simplex_geotrans(N,1)` は次元 N のシンプレックスのための通常の線形幾何変換を示します。

シンプレックスのために、以下のようにより特殊化された関数が存在します。

```
mymesh.add_simplex(2, ind.begin());
```

次の関数を使ってポイントのリストを直接与えることも可能です。

```
mymesh.add_convex_by_points(pgt, itp);
```

ここで `itp` は点の配列のイテレータです。例えば、次のような場合、

```
std::vector<bgeot::base_node> pts(3);
pts[0] = bgeot::base_node(0.0, 0.0, 0.0);
pts[1] = bgeot::base_node(0.0, 1.0, 0.0);
pts[2] = bgeot::base_node(0.0, 0.0, 1.0);
mymesh.add_convex_by_points(bgeot::simplex_geotrans(2,1), pts.begin());
```

このように使用することも可能です。

```
mymesh.add_simplex_by_points(2, pts.begin());
```

シンプレックス以外の要素についても、`mymesh.add_convex_by_points` や `mymesh.add_convex` で適切な幾何学的変換と共に使用することが可能です。

- `bgeot::parallelepiped_geotrans(N, 1)` は次元 N の平行 6 面体 ($N = 2$ の場合は四辺形、 $N = 3$ の場合は 6 面体、...)
- `bgeot::prism_geotrans(N, 1)` は次元 N のプリズムの通常の変換を表しています (通常のプリズムは $N = 3$ です。一般化されたプリズムは、次元 $N-1$ のシンプレックスとセグメント)

特殊化された関数も存在します

```
mymesh.add_parallelepiped(N, it);
mymesh.add_parallelepiped_by_points(N, itp);
mymesh.add_prism(N, it);
mymesh.add_prism_by_points(N, itp);
```

点の配列における点の順序は、シンプレックス要素にとっては重要ではありません (あなたがシンプレックスの向きを気にする場合を除いては)。他の要素については、**通常の一次要素の頂点数** (一次要素) で示される頂点の順序を守ることが重要です。

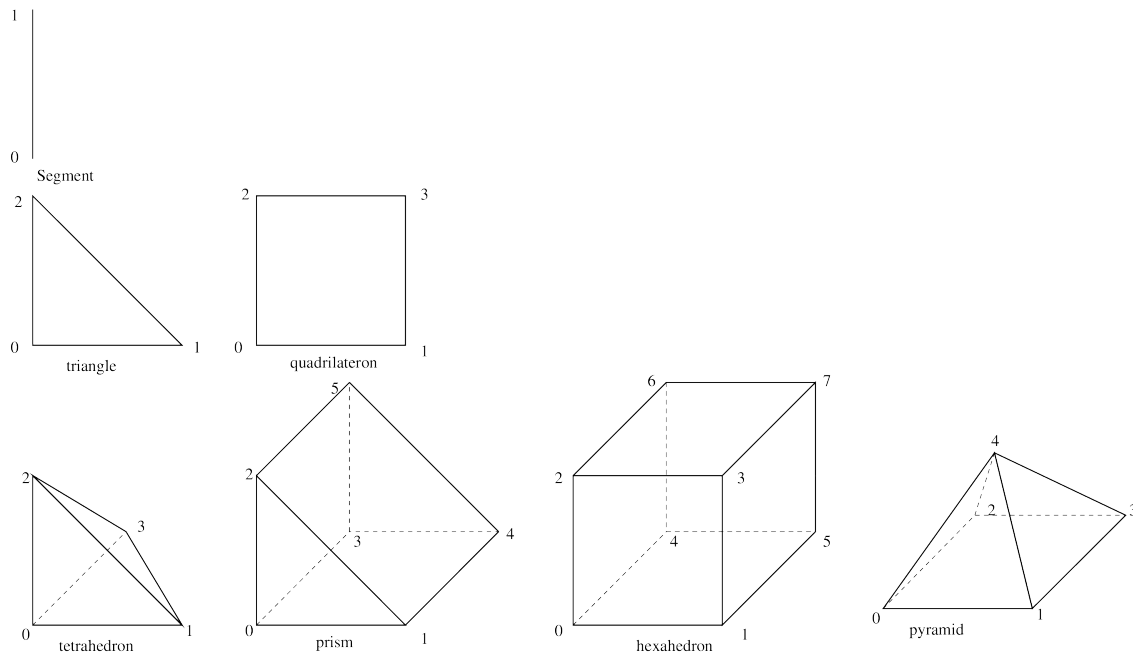


図 6.1 通常の一次要素の頂点数

より高次の変換を含む汎用的な規則は、頂点数が対応する **Lagrange 有限要素法** (付録 A. 有限要素法リスト を参照) の 1 つに従うことであることに注意してください。

6.2 メッシュから要素を削除する

メッシュから要素を削除するには、単純に

```
mymesh.sup_convex(i);
```

ここで、`i` は要素のインデックスです。

6.3 シンプルな構造化メッシュ

平行 6 面体の領域の場合、`getfem/getfem_regular_meshes.h` で定義されている 3 つの関数からシンプルレックス、平行 6 面体、またはプリズム要素を持つ構造化メッシュを得ることができます。

最も単純な関数は

```
void regular_unit_mesh(mesh& m, std::vector<size_type> nsubdiv,
                      bgeot::pgeometric_trans pgt, bool noised = false);
```

メッシュ `m` をシンプル/平行 6 面体/プリズムの規則的なメッシュで塗りつぶします (`pgt` の値に応じて)。各方向のセル数は `nsubdiv` で与えられます。次の例では、単位正方形に 2 次 3 角形のメッシュを作成します (メッシュは後でスケールして変換できます)

```
std::vector<getfem::size_type> nsubdiv(2);
nsubdiv[0] = 10; nsubdiv[1] = 20;
regular_unit_mesh(m, nsubdiv, bgeot::simplex_geotrans(2,2));
```

もっと特殊な規則的なメッシュ関数も利用できます

```
getfem::parallelepiped_regular_simplex_mesh(mymesh, N, org, ivect, iref);
getfem::parallelepiped_regular_prism_mesh(mymesh, N, org, ivect, iref);
getfem::parallelepiped_regular_pyramid_mesh(mymesh, N, org, ivect, iref);
getfem::parallelepiped_regular_mesh(mymesh, N, org, ivect, iref);
```

ここで `mymesh` は構造化されたメッシュが構築されるメッシュ変数で、`N` は次元です (シンプルレックスの場合は 4、プリズムの場合は 5、平行 6 面体の場合は無制限です)。`org` を `bgeot::base_node` と設定し、メッシュの起点を表します。`ivect` は平行四辺形の領域を構築するための `N` ベクトルの配列上のイテレータです。`iref` は、各方向の分割数を表す `N` の自然数の配列です。

例えば、 $10 \times 10 \times 10$ の単位立方体の 4 面体を持つメッシュを作成するには、次のように記述します

```
getfem::mesh mymesh;
bgeot::base_node org(0.0, 0.0, 0.0);
std::vector<bgeot::base_small_vector> vect(3);
vect[0] = bgeot::base_small_vector(0.1, 0.0, 0.0);
```

```

vect[1] = bgeot::base_small_vector(0.0, 0.1, 0.0);
vect[2] = bgeot::base_small_vector(0.0, 0.0, 0.1);
std::vector<int> ref(3);
ref[0] = ref[1] = ref[2] = 10;
getfem::parallelepipiped_regular_simplex_mesh(mymesh, 3, org, vect.begin(), ref.begin());

```

ノート: `base_node` と `base_small_vector` はどちらも小さなベクトルクラス (16 個以上の要素を格納できません) で、幾何用の点や幾何用のベクトルを記述するために使われます。それらのメモリフットプリントは `std::vector` よりも小さくなっています。

6.4 メッシュ領域

メッシュオブジェクトには、多くの `getfem::mesh_region` オブジェクト (`getfem/getfem_mesh_region.h` で宣言) を含めることができます。これらのオブジェクトは、凸と凸面のセットのコンテナです。それらは、境界を定義するため、または並列ソルバーのメッシュの区画などに使用されます。

```

mymesh.region(30).add(2); // adds convex 2 into region 30
mymesh.region(30).add(3); // adds convex 3 into region 30
mymesh.region(30).add(4,3); // adds face 3 of convex 4 into region 30
mymesh.region(30).sup(3); // Removes convex 3 from region 30
mymesh.sup_convex(4); // Removes convex 4 from both the mesh and all the regions
for (getfem::mr_visitor i(mymesh.region(30)); !i.finished(); ++i) {
    cout << "convex: " << i.cv() << " face:" << i.f() << endl;
}

```

6.5 `getfem::mesh` オブジェクトのメソッド

リストは網羅的ではありません。

`mymesh.dim()`

メッシュの主次元。

`mymesh.points_index()`

メッシュの有効なポイントのすべてのインデックスを表す `dal::bit_vector` オブジェクトを返します (下記参照)。

`mymesh.points()[i]`

インデックス `i` のポイントを与えます (`bgeot::base_node`)。

`mymesh.convex_index()`

メッシュの有効な要素のすべてのインデックスを表す “`dal::bit_vector`” オブジェクトを返します (下記参照)。

`mymesh.structure_of_convex(i)`

インデックス i の要素の構造の記述を与えます。この関数は `bgeot::pconvex_structure` を返します。

`mymesh.structure_of_convex(i)->nb_faces()`

インデックス i の要素の面の数。

`mymesh.structure_of_convex(i)->nb_points()`

インデックス i の要素の頂点の数。

`mymesh.structure_of_convex(i)->dim()`

インデックス i の要素の内在次元。

`mymesh.structure_of_convex(i)->nb_points_of_face(f)`

インデックス i の要素のローカルインデックス f の面の頂点の数。

`mymesh.structure_of_convex(i)->ind_points_of_face(f)`

インデックス i の要素のローカルインデックス f の面のすべての頂点のローカルインデックスを持つコンテナを返します。例えば、`mesh.structure_of_convex(i)->ind_points_of_face(f)[0]` は最初の頂点のローカルインデックスです。

`mymesh.structure_of_convex(i)->face_structure(f)`

インデックス i の要素のローカルインデックス f の構造 (`bgeot::pconvex_structure`) を与えます。

`mymesh.ind_points_of_convex(i)`

インデックス i の要素の頂点のグローバルインデックスを持つコンテナを返します。

`mymesh.points_of_convex(i)`

インデックス i の要素の頂点を持つコンテナを返します。これは `bgeot::base_node` の配列です。

`mymesh.convex_to_point(ipt)`

グローバルインデックス ipt のポイントにアタッチされたすべての要素のインデックスを持つコンテナを返します。

`mymesh.neighbours_of_convex(ic,f)`

要素 ic 以外の要素 ic のローカルインデックス f に共通面を持つ mesh 内のすべての要素のインデックスを持つコンテナを返します。

`mymesh.neighbour_of_convex(ic,f)`

要素 `ic` 以外の要素 `ic` のローカルインデックス `f` の共通面を持つ `mesh` の最初の要素のインデックスを返します。見つからなければ `size_type (-1)` を返します。

`mymesh.is_convex_having_neighbour (ic, f)`

要素 `ic` がローカルインデックス `f` の面に対して隣接要素を持っているかどうかを返します。

`mymesh.clear ()`

メッシュからすべての要素とポイントを削除します。

`mymesh.optimize_structure ()`

構造体をコンパクトにします (番号付けに穴がないようにポイントと凸包の番号を付け直します)。

`mymesh.trans_of_convex (i)`

インデックス `i` の要素 (`bgeot::pgeometric_trans`) の幾何学的変換を返します。幾何学的変換の詳細については、`dp` を参照してください。

`mymesh.normal_of_face_of_convex (ic, f, pt)`

ローカル座標 (参照要素内の座標) のポイントでローカルインデックス `f` の面で要素の外向き法線を表す `bgeot::base_small_vector` を返します。幾何変換が線形の場合、点 `pt` は影響を与えません。これは単位法線ではなく、結果として生じるベクトルのノルムは、基準要素の面の表面と実要素の面の表面との間の比です。

`mymesh.convex_area_estimate (ic)`

`ic` 凸面の面積の推定値を与えます。

`mymesh.convex_quality_estimate (ic)`

要素 `ic` の品質の概算を与えます。

`mymesh.convex_radius_estimate (ic)`

要素 `ic` の半径の推定値を与えます。

`mymesh.region (irg)`

`getfem::mesh_region` を返します。領域はメッシュに格納され、一連の凸状の番号と凸状の面を含むことができます。

`mymesh.has_region (irg)`

インデックス `irg` が作成されている場合は `true` を返します。

凸/凸面コンテナ `getfem::mesh_region` のメソッドは次のとおりです :

add (ic)

その領域にインデックス “`ic`” の凸包を追加します。

add (ic, f)

“`ic`” 番目の凸の面番号 `f` を追加してください。

sup (*ic*)

sup (*ic*,*f*)

その領域から凸または凸面を除去します。

is_in (*ic*)

is_in (*ic*,*f*)

凸 (または凸面) が領域内にある場合は **true** を返します。

is_only_faces ()

領域に凸が含まれていない場合は **true** を返します。

is_only_convexes ()

領域に凸面が含まれていない場合は **true** を返します。

index ()

領域に格納されている (または面が格納されている) 凸のリストを含む `dal::bit_vector` を返します。

`getfem::mesh_region` に対する反復は、`getfem::mr_visitor` で行う必要があります

```
getfem::mesh_region &rg = mymesh.region(2);
for (getfem::mr_visitor i(rg); !i.finished(); ++i) {
    cout << "contains convex " << i.cv();
    if (i.is_face()) cout << "face " << i.f() << endl;
}
```

6.6 `dal::bit_vector` を使います

オブジェクト `dal::bit_vector` (`getfem/dal_bit_vector.h` で宣言されています) は、*GetFEM++* で頻繁に使用される構造体です。これは `std::bitset` と `std::vector<bool>` に非常に近いものですが、非負の整数の集合を表現する追加の機能を持ち、それらを繰り返し処理します。

`nn` が `dal::bit_vector` であると宣言されている場合、2つの命令 `nn.add(6)` または `nn[6] = true` は等価であり、整数 6 がその集合に追加されることを意味します。

同様に `nn.sup(6)` または `nn[6] = false` で整数 6 を集合から削除します。命令 `nn.add(6,4)` は集合に 6,7,8,9 を追加します。

`dal::bit_vector` を反復するにはイテレーターを通常どおり使用することができますが、ほとんどの場合、このオブジェクトは整数の集合を表しているため、集合に含まれる整数を反復したいだけです。これを行う最も簡単な方法は、疑似反復子 `dal::bv_visitor` を使用することです。

例えば、メッシュ上の点を標準出力に出力するコードを次に示します

```
for (dal::bv_visitor i(mymesh.points_index()); !i.finished(); ++i)
  cout << "Point of index " << i << " of the mesh: " << mymesh.points()[i] << endl;
```

6.7 面番号

通常の要素の面数を図 通常の要素の面に対するの記数 に示します。

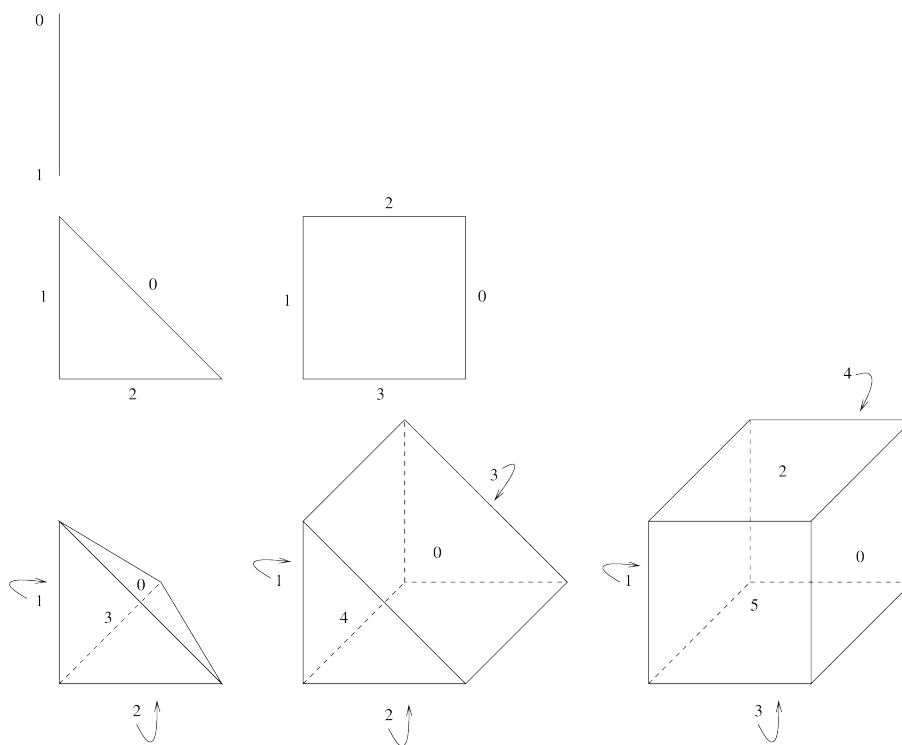


図 6.2 通常要素の面に対するの記数

凸と点は `gf_m` オブジェクト内でグローバルに番号が付けられていますが、面のグローバルな番号付けはありません。そのため、与えられた面を参照する唯一の方法は凸面の番号と凸面の局所面番号を与えますことです。

6.8 メッシュの保存とロード

6.8.1 GetFEM++ ファイル形式から

`getfem / getfem_mesh.h` には、ファイルからメッシュをロードし、メッシュをファイルに書き込むための2つのメソッドが定義されています。

`mymesh.write_to_file` (*const std::string &name*)

メッシュをファイルに保存します。

`mymesh.read_from_file` (*const std::string &name*)

ファイルからメッシュを読み込みます。

メッシュを読み込んで情報を抽出する方法の例を以下に示します

```
#include <getfem/getfem_mesh.h>
```

```
getfem::mesh mymesh;
```

```
int main(int argc, char *argv[]) {
  try {
    // read the mesh from the file name given by the first argument
    mymesh.read_from_file(std::string(argv[1]));

    // List all the convexes
    dal::bit_vector nn = mymesh.convex_index();
    bgeot::size_type i;
    for (i << nn; i != bgeot::size_type(-1); i << nn) {
      cout << "Convex of index " << i << endl;
      bgeot::pconvex_structure cvs = mymesh.structure_of_convex(i);
      cout << "Number of vertices: " << cvs->nb_points() << endl;
      cout << "Number of faces: " << cvs->nb_faces() << endl;
      for (bgeot::short_type f = 0; f < cvs->nb_faces(); ++f) {
        cout << "face " << f << " has " << cvs->nb_points_of_face(f);
        cout << " vertices with local indexes: ";
        for (bgeot::size_type k = 0; k < cvs->nb_points_of_face(f); ++k)
          cout << cvs->ind_points_of_face(f)[k] << " ";
        cout << " and global indexes: ";
        for (bgeot::size_type k = 0; k < cvs->nb_points_of_face(f); ++k)
          cout << mymesh.ind_points_of_convex(i)[cvs->ind_points_of_face(f)[k]] << " ";
      }
    }
  }
} GMM_STANDARD_CATCH_ERROR; // catches standard errors
```

6.8.2 メッシュをインポートする

`getfem/getfem_import.h` は次の関数を提供します:

```
void import_mesh(const std::string& fmtfilename, mesh& m);
```

ここで、文字列 `fmtfilename` には、ファイル形式 (“gid”, “gmsh”, “cdb”, “noboite”, “am_fmt”, “emc2_mesh”,

または”structured”)の記述子と、その後にはコロンとファイル名が含まれていなければなりません(書式記述子がない場合、そのファイルはネイティブの `getfem` メッシュであり、`mesh::read_from_file()` メソッドが使用されているものとみなされます)。例えば:

```
getfem::mesh m;
getfem::import_mesh("gid:../tests/meshes/tripod.GiD.msh",m);
```

あるいは、関数

```
void import_mesh(const std::string& filename, const std::string& fmt,
                mesh& m);
```

前述の書式指定子の1つである文字列 `fmt` と同等の方法で使用できます。

“gid”フォーマット指定子は、`GiD`によって生成されるメッシュ用であり、`gmsch`は、オープンソースメッシュ生成プログラム `Gmsh`によって生成されるメッシュ用です。“cdb”書式指定子は `CDWRITE` コマンドでブロック形式で出力された `ANSYS` モデルからメッシュを読み込むためのものです。現在、`ANSYS` 要素タイプ 42,45,73,82,87,89,90,92,95,162,182,183,185,186,187、および 191 をインポートすることができますが、これらの要素にリンクされた有限要素技術は含まず、そのジオメトリのみが含まれます。“noboite”形式は `TetMesh-GHS3D` 用で、“am_fmt”と“emc2_mesh”は“EMC2”(ただし 2D のみ)で構築されたファイル用です。

構造化形式は通常のメッシュの短い仕様です。その場合の `fmtfilename` の残りの部分はファイル名ではなく、次の形式の文字列です

```
getfem::import_mesh("structured:GT='GT_PK(2,1)';"
                    "NSUBDIV=[5,5];"
                    "ORG=[0,0];"
                    "SIZES=[1,1];"
                    "NOISED=0", m);
```

ここで `GT` は幾何変換の名前、`NSUBDIV` は各座標(デフォルト値 2)の細分数のベクトル、`ORG` はメッシュの原点(デフォルト値 `[0,0,...]`)、`SIZES` は各方向のサイズのベクトル(デフォルト値 `[1, 1, ...]`)、`NOISED = 1` の場合は、メッシュ内部の節点がランダムに振られます(デフォルト値 `NOISED = 0`)。GT 以外のパラメータはオプションです。

第 7 章

メッシュ上に有限要素法を構築します

`getfem/getfem_mesh_fem.h` で定義されているオブジェクト `getfem::mesh_fem` は、メッシュ全体に有限要素法を記述するために、すなわちいくつかの変数が記述される有限要素空間を記述するために設計されています。これは *GetFEM++* の中心にあるかなり複雑なオブジェクトです。基本的に、この構造は、メッシュの各要素の有限要素法と、いくつかの追加のオプションの変換を記述します。1つのメッシュに対して任意の数の有限要素法を持つことが可能です。これは、混合メソッドには特に必要ですが、同じメッシュ上の異なるデータを記述する場合にも必要です。次のように `getfem::mesh_fem` オブジェクトをインスタンス化することもできます:

```
getfem::mesh_fem mf(mymesh);
```

`mymesh` は既に存在するメッシュです。構造はこのメッシュにリンクされ、メッシュに変更が行われると同期されます。

要素ごとに有限要素法を指定することができるので、たとえ次元が異なっても混合型要素を扱うことができます。通常の要素では、2つの要素が互換性がある場合（共通面で同じ自由度）、2つの要素の間の接続が行われません。自由度の数値は、Cuthill Mc Kee アルゴリズムのように自動的に行われます。メッシュの頂点の数と自由度の数値との間には絶対的な関係はないことに注意してください。すべての `getfem::mesh_fem` オブジェクトには独自の数値があります。

`getfem::mesh_fem` オブジェクトには3つのレベルがあります。

- 要素レベル：要素ごとに1つの有限要素法。要素の次元とベクトルまたはスカラーのプロパティを混在させることは可能です。
- オプションのベクトル化/テンソル化 (`getfem` の専門用語 `qdim`、`vocabulary` を参照)。例えば、連続体力学における変位またはテンソル場を表します。スカラー要素は成分単位で使用されます。ベクトル化されない固有のベクトル要素（例えば、Raviart-Thomas 要素）とスカラー要素を混在させることができます。
- (*GetFEM++* version 4.0) 自由度のオプションの追加線形変換（縮小）。これは、2つの行列すなわち、縮小行列と拡張行列を与えます。リダクション行列は、基本的な自由度を縮小された自由度に変換する必要

があります (縮小された自由度の数は、基本的な自由度の数よりも少ないか等しい必要があります)。拡張行列は、逆変換を記述する必要があります。縮小行列と拡張行列との積は、単位行列でなければなりません (特に、2つの行列が最大ランクであることを保証する必要があります)。この任意の変換を使用して、有限要素空間を特定の領域 (基本的に境界) に縮小したり、本来適合しない fem (例えば、自由度の異なる fem) 間のいくつかの一致条件を指定することができます。

`getfem::mesh_fem` オブジェクトの自由度を操作するこの構造を念頭に置いておく必要があります。

7.1 第 1 レベル : 各要素の fem の操作

特定の要素に対して特定の有限要素法を選択するメソッドは:

```
mf.set_finite_element(i, pf);
```

`i` は要素のインデックスで、`pf` は有限要素法の記述子 (`getfem::pfem` 型、基本的には `getfem::virtual_fem` から継承するオブジェクトへのポインタ) です。このメンバ関数の代わりに形式は:

```
void mesh_fem::set_finite_element(const dal::bit_vector &cvs,
                                  getfem::pfem pf);
void mesh_fem::set_finite_element(getfem::pfem pf);
```

`bit_vector cvs` にリストアップされた凸包、またはメッシュのすべての凸包の有限要素を設定します。最後のメソッドは次のメソッドを呼び出します:

```
void mesh_fem::set_auto_add(pfem pf);
```

これは、メッシュの新しい要素に自動的に追加されるデフォルトの有限要素法を定義します (例えば、メッシュの細分化が実行されるときに非常に便利です)。

有限要素法と積分法の記述子は、次の関数によって利用できます。

```
getfem::pfem pf = getfem::fem_descriptor("name of method");
```

既存のメソッドの中で "メソッド名" を選択する場所です。メソッドの名前は、次の関数によって取得できます:

```
std::string femname = getfem::name_of_fem(pf);
```

有限要素法の非網羅的リスト (付録 A. 有限要素法リスト または `getfem/getfem_fem.h` を参照) は次のように与えられます:

- "FEM_PK(n, k)": 次数 k の多項式を用いた次元 n の古典的な単項の P_K 法
- "FEM_QK(n, k)": 次元 n の平行 6 面体上の古典的 Q_K 法。セグメント上の次数 k の P_K 法のテンソルによる生成物。

- "FEM_PK_PRISM(n, k)": 次元 n のプリズムに関する古典的な方法。2 次の k の $|P_K|$ 法のテンソル積。
- "FEM_PRODUCT(a, b)": 2 つの多項式有限要素法 a と b のテンソル積。
- "FEM_PK_DISCONTINUOUS(n, k)": 次数 k の多項式を持つ次元 n のシンプレックスに関する不連続 P_K 法。

与えられた幾何学変換に適した Lagrange 多項式の fem を得る別の方法は:

```
getfem::pfem getfem::classical_fem(bgeot::pgeometric_trans pg,
                                   short_type degree);
getfem::pfem getfem::classical_discontinuous_fem(bgeot::pgeometric_trans pg,
                                                  short_type degree);
```

`mesh_fem` は次のように直接これらの関数を呼び出すことができます

```
void mesh_fem::set_classical_finite_element(const dal::bit_vector &cvs,
                                             dim_type fem_degree);
void mesh_fem::set_classical_discontinuous_finite_element(const dal::bit_vector &cvs,
                                                           dim_type fem_degree);
void mesh_fem::set_classical_finite_element(dim_type fem_degree);
void mesh_fem::set_classical_discontinuous_finite_element(dim_type fem_degree);
```

いくつかの他の方法:

`mf.convex_index()`

有限要素法が定義されている索引 (`dal::bit_vector`) の集合。

`mf.linked_mesh()`

リンクされたメッシュへの参照を与えます。

`mf.fem_of_element(i)`

インデックス “ i ” の要素に定義されている有限要素法の記述子を与えます (`qdim` もオプションの削減も考慮しない)。

`mf.clear()`

構造をクリアしており、有限要素法はまだ定義されていません。

7.2 例

例えば、3 角形上に P_1 有限要素法の記述が必要な場合、それを設定する方法は:

```
mf.set_finite_element(i, getfem::fem_descriptor("FEM_PK(2, 1)"));
```

ここで i はまだ 3 角形のインデックスです。単一のインデックスではなく、`mf.set_finite_element` に `dal::bit_vector` を渡して、要素セット上で直接特定のメソッドを選択することもできます。例えば:

```
mf.set_finite_element(mymesh.convex_index(),
    getfem::fem_descriptor("FEM_PK(2, 1)"));
```

はメッシュのすべての要素のメソッドを選択します。

7.3 第 2 レベル : オプションである「ベクトル化/テンソル化」

有限要素がベクトルフィールドである未知数を表す場合、メソッド `mf.set_qdim(Q)` は、ターゲットディメンション Q の定義のターゲットディメンションを設定します。

対象の次元 Q が 1 以外の値に設定されている場合、スカラ FEM (P_k fem など) は `mesh_fem` オブジェクトの観点から自動的に「ベクトル化」されます。つまり、各スカラー自由度はベクトル場の Q 成分を表現するため Q 回表示されます。本質的にベクトル要素が使用される場合、fem のターゲットディメンションと `mesh_fem` オブジェクトのターゲットディメンションが一致しなければなりません。それを要約すると、

- i の要素の fem が本質的にベクトル FEM である場合:

```
mf.get_qdim() == mf.fem_of_element(i)->target_dim()
&&
mf.nb_dof_of_element(i) == mf.fem_of_element(i).nb_dof()
```

- もし FEM が 1 に等しい target_dim を持つならば:

```
mf.nb_dof_of_element(i) == mf.get_qdim()*mf.fem_of_element(i).nb_dof()
```

さらに、表現されるフィールドがベクトル場の代わりにテンソル場 (例えば、弾性の応力またはひずみテンソル場) である場合、テンソルの次元を以下の方法で指定することが可能です。

```
mf.set_qdim(dim_type M, dim_type N)
mf.set_qdim(dim_type M, dim_type N, dim_type O, dim_type P)
mf.set_qdim(const bgeot::multi_index &mii)
```

2 次、4 次、任意 (但し、6 に限定される) のテンソル場について、ほとんどの演算では、これは次元の積であるベクトル場を宣言するのと同じです。しかし、宣言されたテンソルの次元は、高水準のジェネリック構築に考慮されます。テンソル内の成分は Fortran の次数で格納されることに注意してください。

このレベルでは、基本的な自由度が定義されています。 `getfem::mesh_fem` のいくつかのメソッドは基本的な自由度に関する情報を得ることができます :

```
mf.nb_basic_dof_of_element(i)
```

インデックス i の要素の基本自由度の数を返します。

`mf.ind_basic_dof_of_element(i)`

インデックス i の要素の基本自由度のすべてのグローバルインデックスを持つコンテナ (配列) を返します。

`mf.point_of_basic_dof(i,j)`

インデックス i の要素上のローカルインデックス j の基本自由度点を表す `bgeot::base_node` を返します。

`mf.point_of_basic_dof(j)`

グローバルインデックス j の基本自由度に関連する点を表す `bgeot::base_node` を返します。

`mf.reference_point_of_basic_dof(i,j)`

参照要素の座標におけるインデックス i の要素上のローカルインデックス j の基本自由度に関連する点を表す `bgeot::base_node` を与えます。

`mf.first_convex_of_basic_dof(j)`

グローバルインデックス j の基本自由度が定義されている最初の要素のインデックスを返します。

`mf.nb_basic_dof()`

異なる基本自由度の総数を与えます。

`mf.get_qdim()`

対象の次元 “Q” を与えます。

`mf.basic_dof_on_region(i)`

凸要素の集合またはインデックス i の面の集合にある基本自由度のインデックスを表す `dal::bit_vector` を返します (`getfem::mesh` オブジェクトを参照)。

`mf.dof_on_region(i)`

凸の集合またはインデックス i の面の集合にある自由度のインデックスを表す `dal::bit_vector` を返します (`getfem::mesh` オブジェクトを参照)。 `mesh_fem` を小さくすると、対応している試行関数がこの領域上でゼロでない場合、自由度は領域上に設定されます。拡張行列は、基本と縮小の自由度間の対応付けを行うために使用されます。

7.4 第 3 レベル : オプションである線形 (または縮小) 変換

上述のように、自由度の線形変換を記述する 2 つの行列、縮小行列 R および拡張行列 E が提供されています。 V が基本自由度のベクトルである場合、 $U = RV$ は上述のように減少した自由度のベクトルとなります。逆に、減少した自由度のベクトル U が与えられると、 $V = EU$ は元の自由度のベクトルとなります。単純な場合、 E は単に R の転置になります。拡張行列のすべての行はスパースでなければなりません。これに注意しないと、各マトリックスの構築は難しくなります!

自然条件として $RE = I$ であり I は単位行列です。

`mf.nb_dof()`

異なる自由度の総数を与えます。オプションのリダクションが使用される場合、これは削減行列の列の数になります。それ以外の場合、基本自由度の数が返されます。

`mf.is_reduced()`

ブール値を返します。リダクションが使用されている場合は True です。

`mf.reduction_matrix()`

縮小行列 R に `const` 参照を返します。

`mf.extension_matrix()`

縮小行列 E に `const` 参照を返す。

`mf.set_reduction_matrices(R, E)`

縮小行列と拡張行列を R と E と定義し、それらの使用法を検証します。

`mf.set_reduction(b)`

b はブール値です。 b が偽であれば削減を取り消し、 b が真であれば削除を検証します。 b が真の場合は、事前に拡張行列と縮約行列を設定している必要があります。

`mf.reduce_to_basic_dof(idof)`

`idof` に存在する基本自由度のみを保持するように対応する縮小行列と拡張行列を設定します。パラメータ `idof` は `dal::bit_vector` または `std::set<size_type>` のいずれかです。これは `getfem::partial_mesh_fem` オブジェクトの使用方法と同じです。

7.5 汎用的な *mesh_fem* の取得

次の関数を使うことができます:

```
const mesh_fem &getfem::classical_mesh_fem(const getfem::mesh &mymesh, dim_type K);
```

与えられた `mymesh` 上で次数 K の古典的多項式 *mesh_fem* を得ることができます。*mesh_fem* はリンクされたメッシュが破棄されると自動的に破棄されます。この関数によって構築されたすべての *mesh_fem* はキャッシュに格納されます。つまり、この関数を同じ引数で 2 回呼び出すと、同じ *mesh_fem* オブジェクトが返されます。つまり、決してこの *mesh_fem* を修正してはいけません!

7.6 `partial_mesh_fem` オブジェクト

`getfem::partial_mesh_fem.h` ファイルで定義された `getfem::partial_mesh_fem` オブジェクトは `getfem::mesh_fem` オブジェクトを自由度の集合に減らすことができます。注目してほしいことは、これは有限要素法の完全な記述ではなく、元の `getfem::mesh_fem` を参照して、縮小と拡張行列を追加しているだけということです。例えば、`getfem::classical_mesh_fem(mesh, K)` 関数で得られた `mesh_fem` を縮約し、メッシュ領域（境界でもよい）上で有限要素法を得ることができます。 `getfem::partial_mesh_fem` は特に指定された境界条件の乗数記述を得るために使用されます。

`getfem::partial_mesh_fem` オブジェクトの宣言は次のとおりです:

```
getfem::partial_mesh_fem partial_mf(mf);
```

次に、適合メソッドを次のように呼び出す必要があります:

```
partial_mf.adapt(kept_dof, rejected_elt = dal::bit_vector());
```

`kept_dof` と `rejected_elt` はいくつかの `dal::bit_vector` です。 `kept_dof` は、保持される元の `mesh_fem` `mf` の自由度インデックスのリストです。 `rejected_elt` は `getfem::partial_mesh_fem` が有限要素法がないことを示す要素インデックスのリストを含むオプションのパラメータです。これは、構築中の不要な計算を避けるためです。

第 8 章

積分方法の選択

メッシュ全体に対する積分方法の設定は、ファイル `getfem/getfem_mesh_im.h` で定義されている構造体 `getfem::mesh_im` のおかげで行われます。基本的に、この構造はメッシュの各要素に対する積分方法を記述します。次のように `getfem::mesh_im` オブジェクトをインスタンス化できます:

```
getfem::mesh_im mim(mymesh);
```

`mymesh` は既に存在するメッシュです。構造はこのメッシュにリンクされ、メッシュが修正されたときに追従します (たとえば、メッシュが洗練された場合、積分法も洗練されます)。

要素ごとに積分方法を指定することができますので、たとえ次元が異なっても混合型の要素を扱うことができます。

特定の要素に対して特定の積分法を選択するには、以下を使用できます:

```
mim.set_integration_method(i, ppi);
```

ここで、`i` は要素のインデックスであり `ppi` は積分方法の記述子です。このメンバ関数の代わりに形式は

```
void mesh_im::set_integration_method(const dal::bit_vector &cvs,
                                     getfem::pintegration_method ppi);
void mesh_im::set_integration_method(getfem::pintegration_method ppi);
```

これは `bit_vector cvs` にリストされた凸包、またはメッシュのすべての凸包のいずれかの積分方法を設定します。

積分法のすべての利用可能な記述子のリストはファイル `:file:getfem/getfem_integration.h` にあります。積分メソッドの記述子は次の関数によって利用可能です:

```
getfem::pintegration_method ppi = getfem::int_method_descriptor("name of method");
```

既存のメソッドの中で "メソッド名" を選択する場所です。メソッドの名前は次のように取り出すことができます:

```
std::string im_name = getfem::name_of_int_method(ppi);
```

積分法の部分的なリスト (付録 B. 立体求積法のリスト または `getfem/getfem_integration.h` を参照) を以下に示します。

正確な積分方法の例:

- "IM_NONE () " : ダミー積分法 (`getfem ++-1.7` の新機能)。
- "IM_EXACT_SIMPLEX (n) " : n 次元の参照の単項上の多項式の正確な積分の描述。
- "IM_PRODUCT (a, b) " : “a” と “b” の凸包の直接の積である凸包の正確な積分の叙述。
- "IM_EXACT_PARALLELEPIPED (n) " : 次元 n の平行 6 面体の多項式の正確な積分の記述です。
- "IM_EXACT_PRISM (n) " : 次元 n の参照のプリズム上の多項式の正確な積分の記述です。

近似された積分方法の例:

- "IM_GAUSS1D (k) " : “k” の次数分の Gauss 積分の記述です。 “99” 以下のすべての奇数値 “k” に対して利用可能です。
- "IM_NC (n, k) " : (Lagrange 補間に基づく) Newton コーツ法を用いて次数 k の多項式の次元 n の単純な積分の記述。
- "IM_PRODUCT (a, b) " : 積分法 a と積分法 b を直積した積分法を構築します。
- "IM_TRIANGLE (2) " : 2 次の 3 角形に 3 点で積分します。
- "IM_TRIANGLE (7) " : 7 次の 3 角形に 13 点で積分します。
- "IM_TRIANGLE (19) " : 19 次の 3 角形に 73 点で積分します。
- "IM_QUAD (2) " : 2 次の 3 角形に 3 点で積分します。
- "IM_GAUSS_PARALLELEPIPED (2, 3) " : 4 点で 3 次の四辺形を積分します ("IM_PRODUCT (IM_GAUSS1D (3), IM_GAUSS1D (3)) " のショートカット)。
- "IM_TETRAHEDRON (5) " : 15 点で 5 次の 4 面体を積分します。

ノート: "IM_QUAD (3) " は, "FEM_QK (2, 3) " 有限要素の基本関数を正確に積分できないことに注意してください。この場合、基本関数は次数 3 の 1 次元多項式のテンソル積であるため、"IM_QUAD (7) " を使用する必要があります (6 は使用できません)。したがって、"IM_GAUSS_PARALLELEPIPED (2, k) " は、積分点が少ないので、"IM_QUAD (2*k) " より常に優先するべきです。

積分方法を得る別の方法:

```
getfem::pintegration_method ppi =  
  getfem::classical_exact_im(bgeot::pgeometric_trans pgt);  
  
getfem::pintegration_method ppi =  
  getfem::classical_approx_im(bgeot::pgeometric_trans pgt, dim_type d);
```

これらの関数は、正確な（すなわち解析的な）積分方法を返すか、または指定された幾何学的変換で定義された凸包について、（少なくとも）”d”以下の次数の多項式を正確に積分することができる近似積分方法を選択します。

8.1 *mesh_im* オブジェクトのメソッド

積分法がメッシュに定義されると、以下の方法で情報を取得することができます（リストは網羅的ではありません）。

mim.convex_index()
積分法が定義されているインデックスのセット（`dal::bit_vector`）。

mim.linked_mesh()
リンクされたメッシュへの参照を与えます。

mim.int_method_of_element(i)
インデックス *i* の要素に定義された積分法に関する記述子を与えます。

mim.clear()
構造をクリアします。メッシュに定義された積分法はなくなります。

第 9 章

メッシュの改善

単純なメッシュ (セグメント、3 角形、4 面体) の場合、1、2 または 3 次元では、Bank et al([bank1983] を参照) の方法によるメッシュの細分化が利用できます。getfem::mesh オブジェクト mymesh の場合、メソッドは:

```
mymesh.Bank_refine (bv);
```

インデックスが bv (dal_bv オブジェクト) に格納されている要素を絞り込みます。メッシュの適合性は、追加の改良 (ここでは緑色の 3 角形) により維持されます。緑色の 3 角形に関する情報 (図 2 次元での Bank の手法の例) はメッシュオブジェクトに格納され、細分化のために集められます ([bank1983] を参照)。

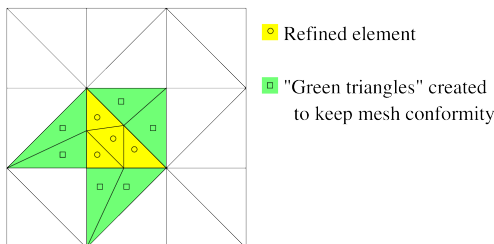


図 9.1 2次元での Bank の手法の例

メッシュの細分化は、大部分が *ポステオリ* なエラーの見積もりと結びついています。とても基本的なエラーの見積もりはファイル getfem/getfem_error_estimate.h 内で利用できます:

```
error_estimate(mim, mf, U, err, rg);
```

ここで、mim は積分法 (getfem::mesh_im オブジェクト)、mf は未知数が計算された有限要素法 (getfem::mesh_fem オブジェクト)、U は未知数の自由度のベクトル、err は誤差推定値が計算されるべき要素 (getfem::mesh_region オブジェクト) からのメッシュ領域塗りつぶしです。

この基本誤差推定は、2 次の問題に対してのみ有効で、各エッジ (2 次元問題の場合) または各面 (3 次元問題の場合) 上の要素間の前後の正規分布の分岐の合計を計算するだけです。これは、メッシュの各面に対して、以下の量

が計算されることを意味しています。

$$\int_e |[[\partial_n u]]|^2 d\Gamma,$$

$[[\partial_n u]]$ は正規微分の分岐です。次に、所与の要素の誤差推定値は、各内面の計算量の合計に要素の直径を掛けたものです。この基本的な誤差の見積もりは、より精巧なもののモデルとみなすことができます。これは、ハイレベルのジェネリック構築と `neighbour_elt` 補間変換を使用します (要素間エッジ/面間の不連続性の評価します。)。

第 10 章

任意の項を計算する - 高水準の汎用的な構築手順 - 弱形式言語

このセクションでは、現在の *GetFEM++* の主要な汎用構築を紹介します。これは、弱微分方程式の境界値問題の弱定式化を記述するために、弱形式言語に基づいているという意味で、高水準の汎用構築です。これは主に、非線形項を記述することが非常に難しい従来の低水準汎用構築（[任意の項を計算する - 低レベルの汎用的な構築手順参照](#)）の問題を回避するために開発されました。逆に、このバージョンでは記号的な微分アルゴリズムが使用されています。これは、非線形結合問題の近似を多く簡略化します。弱形式のみを記述する必要があるため、接線系は自動的に計算されるためです。さらに、弱形式言語は、最適な計算コストを得るために各積分点の評価の前に最適化された命令にコンパイルされます。

C++ で高水準の汎用構築プロシージャを使用するためにインクルードするヘッダーファイルは、`getfem/generic_assembly.h` です。

10.1 高水準と低水準の汎用構築間の実行時間の差

基本的な線形構築の項については、高水準の汎用構築は、低水準の構築よりも高速になります。これは、高水準汎用構築が基本的な最適化された命令にコンパイルを組み込み、単純化を実行するためです。複数の単語を使う場合は、繰り返される項を単純化することにより高速になることがあります。一方、低水準汎用構築が、参照要素に線形変換を伴う要素の線形項を事前計算するメカニズムを組み込んでいるため、いくつかの単純な線形項でより速くなります。もちろん、高水準汎用構築における線形変換のための線形項を基準要素にあらかじめ計算する能力を組み込むことも可能です。しかし、言語の汎用性が高いために複雑になると思われます。結果として、高水準汎用構築では正確な積分が許可されないことになります。

10.2 弱形式言語構文の概要

境界値問題の弱定式化を記述するために、特定の弱形式言語が開発されています。これは、標準的な弱定式化の構造に近いことを意図しており、以下の成分を含みます。

- 変数名: 変数のリストを与える必要があります。変数は有限要素法で記述されるか、未知変数の単純なベクトルになります。例えば、`u`、`v`、`p`、`pressure`、`electric_field` は有効な変数名です。
- 定数名: 定数のリストを与えることができます。ルールは変数と同じですが、試行関数を定数に関連付けることはできません。
- 試行関数: 任意の変数に対して使用できます。試行関数は接頭辞 `Test_` とそれに続く対応する変数名によって識別されます。例えば `Test_u`、`Test_v`、`Test_p`、`Test_pressure`、`Test_electric_field` のようなものです。接触 `model` の場合、2 次の試行関数は `Test2_` と表記されその後ろに変数名が続きます。
- 勾配: 変数または試行関数の空間的な勾配は、`Grad_` を変数名か `Test_`、`Test2_` の接頭辞にすることで表現できます。FEM 変数でのみ使用可能です。たとえば、`Grad_u`、`Grad_pressure`、`Grad_electric_field`、および `Grad_Test_u`、`Grad_Test2_v` です。ベクトルの場合、`Div_u` と `Div_Test_u` は、それぞれ `Trace(Grad_u)` と `Trace(Grad_Test_u)` と等価です。
- Hessian: 変数または試行関数の Hessian 行列は、`Hess_` を変数名または `Test_` または `Test2_` の接頭辞にすることで表現できます。FEM 変数でのみ使用可能です。たとえば、`Hess_u`、`Hess_v`、`Hess_p`、`Hess_Test2_v`、`Hess_Test_p`、`Hess_Test_pressure` などです。
- いくつかの定義済みのスカラー関数 (`sin(t)`、`cos(t)`、`pow(t,u)`、`sqrt(t)`、`sqr(t)`、`Heaviside(t)`、...)。スカラー関数は、スカラーまたはベクトル/行列/テンソル式に適用することができます。成分ごとに適用されます。2 つの引数 (`pow(t,u)`、`min(t,u)` ...) を持つ関数の場合、2 つの非スカラー引数が渡された場合、次元は同じでなければなりません。たとえば、“`max([1; 2], [0; 3])`” は “[1;3]” を返します。
- `+`、`-`、`*`、`/`、`:`、`..`、`*`、`./`、`@`、`'` となります。
- いくつかの定数: `pi`、`meshdim` (現在のメッシュの次元)、変数 `u` の大きさ `qdim(u)` と `qdims(u)` (固定サイズの変数のサイズと FEM 変数のベクトルフィールドの次元)、`Id(n)` は $n \times n$ 次の単位行列です。
- 括弧は標準的な方法で操作の順序を変更するために使用できます。例えば `(1+2)*4` や `(u+v)*Test_u` は有効な式です。
- ベクトル/行列/テンソルの成分へのアクセスは、左括弧、成分のリスト、右括弧の項をたどることによって行うことができます。たとえば、`[1,1,2](3)` は正しく 2 を返します。インデックスは 1 で始まることに注意してください (C++ や Python インタフェースでも)。コロンの使うことで Matlab のような構文でインデックス先の値を置き換えることができます。

- 陽なベクトル: 例えば $[1; 2; 3; 4]$ はサイズが4の陽なベクトルです。各成分は式になります。
- 陽行列: たとえば $[1, 3; 2, 4]$ と $[[1, 2], [3, 4]]$ は同じ 2×2 行列を表します。各成分は式になります。
- 陽な4次テンソル: 入れ子形式の陽な $3 \times 2 \times 2 \times 2$ の4次テンソルの例:
 $[[[[1, 2, 3], [1, 2, 3]], [[1, 2, 3], [1, 2, 3]]], [[[[1, 2, 3], [1, 2, 3]], [[1, 2, 3], [1, 2, 3]]]]]$
 です。
- X は実要素上の現在の座標です。 $X(i)$ は i 番目の要素です。
- `Normal` は領域境界に積分するときの境界に対する外向きの単位法線ベクトル、または `mesh_im_level_set` メソッドでレベル集合に積分するときのレベル集合への単位法線ベクトルです。後者の場合、法線ベクトルはレベル集合関数勾配の方向にあります。
- `Reshape(t, i, j, ...)`: ベクトル/行列/テンソルを変形します。 *GetFEM++* のすべてのテンソルは `Fortran` の順序で格納されることに留意してください。
- ある数の線形と非線形の演算子 (`Trace`、`Norm`、`Det`、`Deviator`、**“Contract”**など)。非線形演算子は試行関数には適用できません。
- `Diff(expression, variable)`: 変数に対する陽な微分可能性 (記号による微分)。
- `Diff(expression, variable, direction)`: `direction` 方向の `variable` に対して `expression` の導関数を計算します。
- `Grad(expression)`: 可能であれば、与えられた式の勾配を記号的に導出します。
- マクロ定義の可能性 (`model` 内では、`ga_workspace` オブジェクトまたは構築文字列内で直接)。マクロは、字句解析フェーズでインライン展開された有効な式でなければなりません (数回使用すると、計算はコンパイル段階で自動的に因数分解されます)。
- `Interpolate(variable, transformation)`: 他の要素や別のメッシュ上の同じメッシュ上で変数を補間したり試行関数を補うことができる強力な演算。 `transformation` は、現在の点から補間を実行する点までのマップを記述する `workspace` または `model` オブジェクトによって格納されるオブジェクトです。この機能は、例えば、周期的な条件を規定するため、または異なるメッシュ上に定義された2つの有限要素空間のモルタル行列を計算するため、またはより一般的には流体構造相互作用のような架空の領域法のために使用できます。
- `Elementary_transformation(variable, transformation)`: 要素水準で定義された線形変換を許可します (つまり、`Gauss` 積分点基準では定義できません)。この機能は、主にプレート要素の縮小 (回転 `RT0` などの低水準のベクトル要素への投影) を定義するために追加されました。 `transformation` は、特定の要素の変換を記述する `workspace` や `model` オブジェクトによって格納されるオブジェクトです。
- 二重積分計算またはカーネル/畳み込み/積分積分による2つの変数の結合のための2つの領域の直接積に対する積分の可能です。これにより、次のような表現が可能になります。 Ω_1 上と Ω_2 上の2つ

の領域で $\int_{\Omega_1} \int_{\Omega_2} k(x, y)u(x)v(y)dydx$ 同じにしろ違うにしろ固有のメッシュと積分手法を持つことができる。ここで、 u は Ω_1 上で定義された変数で v は Ω_2 上で定義された変数です。キーワード `Secondary_domain` (変数) を使用すると、積分の第2領域上の変数にアクセスできます。

10.3 いくつかの基本的な例

領域 Ω 上の Poisson 問題の弱定式化。

$$-\text{div } \nabla u = f, \text{ in } \Omega,$$

$\partial\Omega$ 上の Dirichlet の境界条件 $u = 0$ は古典的な表現です

$$\int_{\Omega} \nabla u \cdot \nabla v dx = \int_{\Omega} f v dx,$$

すべての試行関数について v が $\partial\Omega$ から消えます。構築文字列の対応する式は:

```
Grad_u.Grad_Test_u - my_f*Test_u
```

ここで、`my_f` はソース項の式です。今の式が

$$-\text{div } a \nabla u = f, \text{ in } \Omega,$$

a がスカラー係数の場合、対応する構築文字列は

```
a*Grad_u.Grad_Test_u - my_f*Test_u
```

で a がスカラー定数またはスカラー項として宣言されなければなりません。それを陽に記述することも可能です。例えば、問題

$$-\text{div } \sin(x_1 + x_2) \nabla u = f, \text{ in } \Omega,$$

では x_1, x_2 はメッシュ上の座標で次のように表現することができます。

```
sin(X(1)+X(2))*Grad_u.Grad_Test_u - my_f*Test_u
```

別の古典的な方程式は線形弾性です。

$$-\text{div } \sigma(u) = f, \text{ in } \Omega,$$

等価線形弾性の場合には、 u ベクトル場と。 $\sigma(u) = \lambda \text{div } u + \mu(\nabla u + (\nabla u)^T)$ が考えられています。弱定式化の記述に対応する構築文字列は以下のように書くことができます:

```
(lambda*Trace(Grad_u)*Id(qdim(u)) + mu*(Grad_u+Grad_u')):Grad_Test_u - my_f.Test_u
```

または:

```
lambda*Div_u*Div_Test_u + mu*(Grad_u + Grad_u'):Grad_Test_u - my_f.Test_u
```

ここでもまた、いくつかの問題を結合するために、係数 `lambda` と `mu` に定数、あるいはスカラーフィールドや陽表現、あるいは他の変数からの式を与えますことができます。たとえば、係数が温度場に依存する場合、次のように書くことができます。

```
my_f1(theta)*Div_u*Div_Test_u
+ my_f2(theta)*(Grad_u + Grad_u'):Grad_Test_u - my_f.Grad_Test_u
```

ここで、`theta` は、Poisson 方程式の解である温度です。

```
Grad_theta.Grad_Test_theta - my_f*Grad_Test_theta
```

`my_f1` と `my_f2` はいくつかの関数です。その場合、2つの関数 `my_f1` と `my_f2` が線形であっても、結合のために問題は非線形であることに注意してください。

10.4 導出順序と記号による微分

構築文字列の導出順序は自動的に検出されます。試行関数が見つからない場合、次数は 0（ポテンシャルエネルギー）とみなされ、1 次試行関数が見つかった場合、その次数は 1（弱定式化）とみなされ、1 次と 2 次の両方の試行関数が見つかったら、次数は 2（接線系）とみなされます。

構築（次のセクションを参照）を実行するには、次数（0,1 または 2）を指定する必要があります。次数 1 の文字列が提供され、次数 2 の構築が必要な場合は、式の記号微分が行われます。次数 0 の文字列が提供され、次数 1 または 2 の構築が必要な場合も同じです。もちろん、その逆は真ではありません。次数 1 の式が指定され、次数 0 の構築一が予想される場合、積分は行われません。任意の弱定式化はポテンシャルエネルギーから導出する必要はないので、これは一般に必要ではありません。

汎用構築を使用する標準的な方法は、次数 1 の式（すなわち、弱定式化）を与えることです。ポテンシャルエネルギーが存在する場合、それを供給することが可能です。しかし、複雑な表現をもたらす接線系を得るためには 2 回導かれます。非線形問題では、2 次式を直接指定することはできません。その理由は、残差を得るために弱定式化が必要であるからです。したがって、対応する次数 1 の項を持たないで接線項を使用することはできません。

重要な注意事項: 結合した問題については、グローバルポテンシャルはしばしば存在しないことに注意してください。ポテンシャルが直接定義された問題の部分を結合するのが難しい場合があります。これを説明するために、いくつかのパラメータ（たとえば弾性係数）を持つポテンシャルを定義し、その係数が別の変数との変動の変化で構成されている場合、弱定式化はもちろんポテンシャルの導関数ではありません、一般的に意味を持たない係数に関するものです。ポテンシャルの定義が例外でなければならないのはこれが理由です。

10.5 構築のための C++ 呼び出し

汎用構築を使用する最も自然な方法は `model` オブジェクトの汎用構築要素を使用することです（汎用的な構築ブリック節を参照してください）。しかし、高水準汎用構築を単独で使用することも可能です。

汎用構築は `getfem/getfem_generic_assembly.h` で定義されたオブジェクト `getfem::ga_workspace` によって駆動されます。

`getfem::ga_workspace` オブジェクトを定義する方法は2つあります。それは `model` に依存し（モデル記述と基本モデルブリックを参照）次のように定義する必要があります。

```
getfem::ga_workspace workspace(model);
```

`model` はあらかじめ `getfem::model` でオブジェクトを定義しています。その場合、考慮する変数と定数は `model` の1つです。第2の方法は次に示す方法によって独立した `getfem::ga_workspace` オブジェクトを定義する方法です:

```
getfem::ga_workspace workspace;
```

その場合、変数と定数を `workspace` に追加する必要があります。これは以下の方法により実行できます:

```
workspace.add_fem_variable(name, mf, I, V);
```

```
workspace.add_fixed_size_variable(name, I, V);
```

```
workspace.add_fem_constant(name, mf, V);
```

```
workspace.add_fixed_size_constant(name, V);
```

```
workspace.add_im_data(name, imd, V);
```

ここで、`name` は変数/定数名です（次の節の名前の制限を参照）。`mf` は有限要素法を記述する `getfem::mesh_fem` オブジェクトです。`I` は組み立てられたベクトル/行列上の変数の間隔を示す `gmm::sub_interval` クラスのオブジェクトであり、`V` は変数/変数の値である `getfem::base_vector` です。最後のメソッドは、`im_data` オブジェクト `imd` に定義された定数を追加します。これは、`mesh_im` オブジェクトの積分点にスカラー/ベクトル/テンソルフィールド情報を格納することを可能にします。

一度宣言され、一度変数と定数が宣言されると、構築文字列を次のようにして `workspace` に追加することができます。

```
workspace.add_expression("my expression", mim, rg = all_convexes());
```

`mim` に対応するメッシュのオプションの有効な領域が `mf` にある場合、`mf` は `getfem::mesh_im` オブジェクトと `rg` です。

前のセクションで説明したように、文字列の順序が自動的に検出され、対応する正接項を得るために記号的な微分が実行されます。

構築文字列が `workspace` に追加されると、以下を呼び出すことができます:

```
workspace.assembly(order);
```

`order` は 0 (ポテンシャルエネルギー)、1 (残差ベクトル) または 2 (線形問題の正接項または剛性行列) に等しくなければなりません。構築の結果は次のように利用できます:

```
workspace.assembled_potential() // For order = 0
```

```
workspace.assembled_vector() // For order = 1
```

```
workspace.assembled_matrix() // For order = 2
```

デフォルトでは、アセンブルされたポテンシャル、ベクトル、およびマトリックスは、構築の開始時にゼロに初期化されます。ただし、インクリメンタル構築を実行するために、構築ベクトルと行列を外部ベクトルと行列に設定することが可能です (お勧めします)。2つの方法:

```
workspace.set_assembled_vector(getfem::base_vector &V);
```

```
workspace.set_assembled_matrix(getfem::model_real_sparse_matrix &K);
```

そうすることができます。正しい次元のベクトルと行列を与えるように注意してください。

また、メソッド:

```
workspace.clear_expressions();
```

付属のすべての式を取り消すことができ、別の構築で同じ `workspace` を再利用することができます。

汎用構築を Python/Scilab/Matlab インタフェースから呼び出すこともできます。詳細はインタフェースの `gf_asm` コマンドを参照してください。

10.6 C++ 構築の例

第1の例として、Poisson 問題の構築を紹介します

$$-\operatorname{div} \nabla u = f, \text{ in } \Omega,$$

剛性行列は次式で与えられ

$$K_{i,j} = \int_{\Omega} \nabla \varphi_i \cdot \nabla \varphi_j dx,$$

次のコードでアセンブルされます:

```

getfem::ga_workspace workspace;
getfem::size_type nbdof = mf.nb_dof();
getfem::base_vector U(nbdof);
workspace.add_fem_variable("u", mf, gmm::sub_interval(0, nbdof), U);
workspace.add_expression("Grad_u.Grad_Test_u", mim);
getfem::model_real_sparse_matrix K(nbdof, nbdof);
workspace.set_assembled_matrix(K);
workspace.assembly(2);

```

もちろん、mf はすでに宣言された `getfem::mesh_fem` オブジェクトであり、mim はすでに同じメッシュ上に `getfem::mesh_im` オブジェクトとして宣言されています。変数の値は、問題の線形性のために実際に使用するわけではないことに注意してください。使用されない変数の値として `getfem::base_vector(nbdof)` を渡すことができます。また、構築一文字列の結果とまったく同じ結果が得られる 2 つの式は、`"Grad_Test2_u.Grad_Test_u"` (すなわち、2 次式) または `"Norm_sqr(Grad_u)/2"` (すなわち、ポテンシャル)。`"Grad_u.Grad_u/2"` か 2 次元問題のための `"[Grad_u(1), Grad_u(2)].[Grad_Test_u(1), Grad_Test_u(2)]"` などでも同じ結果が得られます。しかしながら、0 次または 2 次の発現を使用する特別な理由がなければ、1 次の表現 (弱定式化) を与えることが推奨されます。

第 2 の例として、連成問題である非圧縮性弾性の混合問題を考えます

$$\begin{aligned}
 -\operatorname{div}(\mu(\nabla u + (\nabla u)^T - pI_d) &= f, \text{ in } \Omega, \\
 \operatorname{div} u &= 0.
 \end{aligned}$$

ここで u はベクトル値の変位であり、 p は圧力です。結合されたシステム全体のマトリックスは以下のように構築します。

```

getfem::ga_workspace workspace;
getfem::size_type nbdofu = mf_u.nb_dof();
getfem::size_type nbdofp = mf_p.nb_dof();
getfem::base_vector U(nbdofu);
getfem::base_vector P(nbdofp);
getfem::base_vector vmu(1); vmu[0] = mu;
workspace.add_fem_variable("u", mf_u, gmm::sub_interval(0, nbdofu), U);
workspace.add_fem_variable("p", mf_p, gmm::sub_interval(nbdofu, nbdofu+nbdofp), P);
workspace.add_fixed_size_constant("mu", vmu);
workspace.add_expression("2*mu*Sym(Grad_u):Grad_Test_u"
    "- p*Trace(Grad_Test_u) - Test_p*Trace(Grad_u)", mim);
getfem::model_real_sparse_matrix K(nbdofu+nbdofp, nbdofu+nbdofp);
workspace.set_assembled_matrix(K);
workspace.assembly(2);

```

ここで `mf_u` と `mf_p` はすでに同じメッシュに定義されている `getfem::mesh_fem` オブジェクトの一部で、`mim` は定義済みの `getfem::mesh_im` オブジェクト、`mu` は Lamé の弾性定数です。このシステムの部分行列の構築を別々に行うことも可能です。

ソース項の構築方法を見てみましょう。体積ソース項の弱定式化は次のとおりです。

$$\int_{\Omega} f v dx$$

ここで、 f はソース項であり、 v は試行関数です。この構築は次のように記述します。

```
getfem::ga_workspace workspace;
getfem::size_type nbdofo = mf_u.nb_dof();
getfem::base_vector U(nbdofo);
workspace.add_fem_variable("u", mf_u, gmm::sub_interval(0, nbdofo), U);
workspace.add_fem_constant("f", mf_data, F);
workspace.add_expression("f*Test_u", mim);
getfem::base_vector L(nbdofo);
workspace.set_assembled_vector(L);
workspace.assembly(1);
```

ソース項が有限要素 `mf_data` とそれに対応する自由度のベクトル `F` で記述されている場合。陽なソース項を定義することも可能です。例えば

```
getfem::ga_workspace workspace;
getfem::size_type nbdofo = mf_u.nb_dof();
getfem::base_vector U(nbdofo);
workspace.add_fem_variable("u", mf_u, gmm::sub_interval(0, nbdofo), U);
workspace.add_expression("sin(X(1)+X(2))*Test_u", mim);
getfem::base_vector L(nbdofo);
workspace.set_assembled_vector(L);
workspace.assembly(1);
```

も有効です。ソース項が境界項 (Neumann 条件の場合) である場合、唯一の違いは、境界に対応するメッシュ領域を次のように与えられなければならないことです。

```
workspace.add_expression("sin(X(1)+X(2))*Test_u", mim, region);
```

ここで `region` はメッシュ領域番号です。

別の例として、単純な非線形弾性問題を説明します。Saint-Venant Kirchhoff 構成則を考えると、次のような基準構成体 Ω 上の弾性エネルギーとみなす必要があります

$$\int_{\Omega} \frac{\lambda}{2} (\text{tr}(E))^2 + \mu \text{tr}(E^2) dx$$

ここで λ, μ は Lamé 定数で E は、式 $E = (\nabla u + (\nabla u)^T + (\nabla u)^T \nabla u) / 2$ で与えられる歪みテンソルです。

この問題では、対応する接線問題を以下のように構築することが可能です。

```

getfem::ga_workspace workspace;
getfem::size_type nbdofu = mf_u.nb_dof();
getfem::base_vector vlamba(1); vlamba[0] = lambda;
getfem::base_vector vmu(1); vmu[0] = mu;
workspace.add_fem_variable("u", mf_u, gmm::sub_interval(0, nbdofu), U);
workspace.add_fixed_size_constant("lambda", vlamba);
workspace.add_fixed_size_constant("mu", vmu);
workspace.add_expression("lambda*sqr(Trace(Grad_u+Grad_u'+Grad_u'*Grad_u))"
    "+ mu*Trace((Grad_u+Grad_u'+Grad_u'*Grad_u)"
    "* (Grad_u+Grad_u'+Grad_u'*Grad_u))", mim);
getfem::base_vector L(nbdofu);
workspace.set_assembled_vector(V);
workspace.assembly(1);
getfem::model_real_sparse_matrix K(nbdofu, nbdofu);
workspace.set_assembled_matrix(K);
workspace.assembly(2);

```

その非線形問題を解くために Newton-Raphson アルゴリズムを適応させます。もちろん、表現はより反復的であり、いくつかの中間非線形演算子を定義することが望ましいでしょう。ただし、繰り返し式は自動的に検出され、構築内で 1 回だけ計算されます。

最後の例は、4 次問題の剛性行列を構築する Kirchhoff-Love 板の問題です。

```

getfem::ga_workspace workspace;
getfem::size_type nbdofu = mf_u.nb_dof();
getfem::base_vector vD(1); vD[0] = D;
getfem::base_vector vnu(1); vnu[0] = nu;
workspace.add_fem_variable("u", mf_u, gmm::sub_interval(0, nbdofu), U);
workspace.add_fixed_size_constant("D", vD);
workspace.add_fixed_size_constant("nu", vnu);
workspace.add_expression("D*(1-nu)*(Hess_u:Hess_Test_u) -"
    "D*nu*Trace(Hess_u)*Trace(Hess_Test_u)", mim);
getfem::model_real_sparse_matrix K(nbdofu, nbdofu);
workspace.set_assembled_matrix(K);
workspace.assembly(2);

```

D は屈曲率であり、nu は Poisson 比です。

10.7 構築言語のスクリプト言語の呼び出し

Python、Scilab、Matlab のインタフェースで使用する場合は、それぞれのドキュメント、特に gf_asm コマンドと model オブジェクトを参照してください。

10.8 テンソル

基本的に、弱形式言語で操作されるのはテンソルです。例えば、スカラー式の 0 次テンソル (例えば、 $3 + \sin(\pi / 2)$)、ベクトル式の 1 テンソル (XX や Grad_u など) スカラー変数)、行列表現の 2 テンソルなどです。効率性の理由から、言語はテンソルを最大 6 まで操作します。この言語は、6 以上の次数のテンソルもサポートするように容易に拡張することができますが、非効率的な計算につながる可能性があります。式に試行関数が含まれている場合 (ベクトルフィールド u の場合は $\text{Trace}(\text{Grad_Test_}u)$ のように)、テンソルは暗黙的に補足成分を持つ各試行関数に対して計算が行われます。これは暗黙のうちに、操作されるテンソルの最大次数は実際には 6 であることを意味します (Grad_Test_u.Grad_Test2_u には 1 次および 2 次試行関数のために 2 つの成分が暗黙に追加されています)。

弾性テンソルを表現するために、または汎用的なベクトル値の未知数の正接項を得るために、4 次のテンソルが必要です。

10.9 変数

変数のリストは `ga_worspace` オブジェクトに (直接または `model` オブジェクトを通して) 与えなければなりません。変数は有限要素法で記述されるか、未知数の単純なベクトルになります。これは、代数方程式を `model` 上の部分方程式に結合することも可能であることを意味します。変数名は、文字 (大文字小文字を区別する) またはアンダースコアの後に文字、数字、またはアンダースコアを続けて指定する必要があります。いくつかの名前は予約されていますが、これは演算子の名前 (Det、Norm、Trace、Deviator など) で、変数名として使用できません。名前は `Test_`、`Test2_`、`Grad_`、`Div_` または `Hess_` で始めることはできません。変数名はあらかじめ定義された関数 (`sin`、`cos`、`acos` ...) と定数 (`pi`、`Normal`、`X`、`Id` ...) を使用することはできません。

10.10 定数またはデータ

定数のリストを `ga_worspace` オブジェクトに与えることもできます。ルールは変数と同じですが、試行関数を定数に関連付けることはできず、定数に関して記号的な微分もありません。スカラー定数は、構成的法則に介入する係数を表すために定義されることが多いです。さらに、定数は、「`im_data`」オブジェクトを介して積分点上に定義されたいくつかのスカラー/ベクトル/テンソルフィールド (例えば、可塑性のような構成法の近似のいくつかの実装のために) です。

10.11 試行関数

各変数は、1次および2次の試行関数に関連付けられています。第1次試行関数は、弱定式化（それが存在すれば潜在的な式を導出する）で使用され、第2次試行関数は接触 `model` で使用されます。変数 `u` に関連する試行関数は `Test_u` と `Test2_u` です。構築文字列は、試行関数に対して線形でなければなりません。構築文字列に `Test_u` という項が存在する結果として、式は変数 `u` に対応する有限要素の各試行関数について評価されます。与えられた要素において、有限要素が試行関数 `N` を持つ場合、`u` がスカラー場である場合、`Test_u` の値は現在の点上の各試行関数の値になります。ゆえに `Test_u` は `N` 値のベクトルの直交面を返します。もちろん、これは言語に暗黙のうちに書かれています。ゆえに、これについて気にする必要はありません。

10.12 勾配

変数または試行関数の勾配は、`Grad_` と変数名または `Test_` とそれに続く変数名によって識別されます。これは、FEM 変数（または定数）でのみ使用できます。例えば、`Grad_u`、`Grad_v`、`Grad_p`、`Grad_pressure`、`Grad_electric_field`、`Grad_Test_u`、`Grad_Test_v`、`Grad_Test_p`、`Grad_Test_pressure`、`Grad_Test_electric_field` です。勾配は、スカラー変数のベクトルかベクトル場変数の行列のいずれかです。後者の場合、第1のインデックスはベクトルフィールド次元に対応し、第2のインデックスは部分導関数のインデックスに対応します。`Div_u` と `Div_Test_u` は、それぞれ `Trace(Grad_u)` と `Trace(Grad_Test_u)` を最適化したショートカットです。

10.13 Hessian

同様に、変数または試行関数のヘッセ行列は、`Hess_` と変数名または `Test_` とそれに続く変数名によって識別されます。これは、FEM 変数でのみ使用できます。例えば、`Hess_u`、`Hess_v`、`Hess_p`、`Hess_pressure`、`Hess_electric_field`、`Hess_Test_u`、`Hess_Test_v`、`Hess_Test_p`、`Hess_Test_pressure`、`Hess_Test_electric_field` です。ヘッセ行列は、スカラー変数の行列またはベクトル場変数の3次元テンソルのいずれかです。後者の場合、第1のインデックスはベクトル場次元に対応し、第2のインデックスは部分導関数に対応します。

10.14 定義済みのスカラー関数

特定の数の事前定義されたスカラー関数を使用することができます。網羅的なリストは以下の通りであり、そのほとんどは対応する C 関数と同等です。

- `sqr(t)` (`t` の平方根、`t*t` に相当)、`pow(t,u)` (`t` の `u` 乗)、`sqrt(t)` (`t` の平方根)、`exp(t)`、`log(t)`、`log10(t)`

- $\sin(t), \cos(t), \tan(t), \operatorname{asin}(t), \operatorname{acos}(t), \operatorname{atan}(t), \operatorname{atan2}(t, u)$
- $\sinh(t), \cosh(t), \tanh(t), \operatorname{asinh}(t), \operatorname{acosh}(t), \operatorname{atanh}(t)$
- $\operatorname{erf}(t), \operatorname{erfc}(t)$
- $\operatorname{sinc}(t)$ (基本的な正弦関数 $\sin(t)/t$)
- $\operatorname{Heaviside}(t)$ (0 for $t < 0, 1$ for $t \geq 0$)
- $\operatorname{sign}(t)$
- $\operatorname{abs}(t)$
- $\operatorname{pos_part}(t)$ ($tH(t)$)
- $\operatorname{reg_pos_part}(t, \operatorname{eps})$ ($(t - \operatorname{eps}/2 - t^2/(2\operatorname{eps}))H(t - \operatorname{eps}) + t^2H(t)/(2\operatorname{eps})$)
- $\operatorname{neg_part}(t)$ ($-tH(-t), \max(t, u), \min(t, u)$)

スカラ関数は、スカラ式に適用することもできますが、テンソル式に適用することもできます。テンソル式に適用される場合、これは成分単位で適用され、結果は同じ次元のテンソルになります。2つの引数 ($\operatorname{pow}(t, u)$, $\operatorname{min}(t, u)$...) を持つ関数の場合、2つの非スカラ引数が渡された場合、次元は同じでなければなりません。たとえば、“ $\operatorname{max}([1; 2], [0; 3])$ ” は “[0; 3]” を返します。

10.15 ユーザー定義のスカラ関数

すでに定義されているスカラ関数にスカラ関数を追加することは可能です。汎用構築は、1つまたは2つのパラメータを持つスカラ関数のみを考慮します。汎用構築にスカラ関数を追加するには、次のように呼び出します。

```
ga_define_function(name, nb_args, expr, der1="", der2="");
```

```
ga_define_function(name, getfem::pscalar_func_onearg f1, der1="");
```

```
ga_define_function(name, getfem::pscalar_func_twoargs f2, der1="", der2="");
```

`name` は定義される関数の名前、`nb_args` は1または2です。最初の呼び出しでは、`expr` は汎用的な弱形式言語で関数を記述する文字列です。最初の変数に `t` を、2番目の変数に `u` (`nb_args` が2に等しい場合) を使います。たとえば、 $\sin(2*t) + \operatorname{sqr}(t)$ は有効な式です。`ga_workspace` オブジェクトで定義された定数やデータを参照することはできません。`der1` と `der2` は `t` と `u` に対する導関数の式です。これらはオプションです。それらが提供されていない場合、導関数が必要な場合には記号的な微分が用いられます。`der1` と `der2` が関数名だけであると定義されている場合、その導関数が対応する関数です。第2の呼び出しでは、`f1` は1つのス

カラーパラメーターを持つスカラー C 関数上の C ポインターでなければならず、`f2` は 2 つのスカラーパラメーターを持つスカラー C 関数上の C ポインターでなければなりません。

さらに、

```
bool ga_function_exists(name)
```

`true` は `name` が既に定義されている関数である場合に返されます。

```
ga_undefine_function(name)
```

関数を再定義することができるように、既に定義済みの関数の定義を取り消します（関数が存在しない場合はアクションはありません）。

10.16 定義されたスカラー関数の導関数

関数名に接頭辞 `Derivative_` を加えることで、定義された関数の導関数を直接参照することができます。たとえば、`Derivative_sin(t)` は `cos(t)` に相当します。`pow(t, u)` のような 2 つの引数の関数の場合、関数名の前に接頭辞 `Derivative_2_` を付けた 2 番目の引数に関して導関数を参照できます。

10.17 バイナリ演算

テンソル間の特定の数のバイナリ演算が利用できます。

- `+` と `-` は、スカラー、ベクトル、行列、またはテンソルの標準的な加減算です。
- `*` はスカラー、行列-ベクトル、行列-行列または（4 次テンソル）-行列の乗法を表します。
- `/` はスカラーによる除算を表します。
- `.` は、ベクトルのスカラー積を表します。より一般的には、ベクトルを持つ最後のインデックスまたは別のテンソルの最初のインデックスを持つテンソルの収縮を表します。`*` と `.` は、行列ベクトルまたは行列と行列の乗算と等価であることに注意してください。
- `:` は Frobenius 行列の積、またはより一般的には、マトリックスを有する 2 つの最後のインデックスに関するテンソルの収縮を表す。`*` と `:` は（4 次テンソル）行列の乗算と等価であることに注意してください。
- `.*` は、2 つのベクトル/行列/テンソルの成分ごとの乗算を表します。
- `./` は、2 つのベクトル/行列/テンソルの成分ごとの除算を表します。
- `@` はテンソル積を表します。

- $\text{Contract}(A, i, B, j)$ は、 A の i 番目のインデックスと B の j 番目のインデックスに関するテンソル A と B の縮小を表します。最初のインデックスは 1 です。例えば $\text{Contract}(V, 1, W, 1)$ は V と W の 2 つのベクトルに対して VW に相当します。
- $\text{Contract}(A, i, j, B, k, l)$ は A のインデックス i, j と B のインデックス k, l に関してテンソル A と B の二重収縮を表します。例えば、 $\text{Contract}(A, 1, 2, B, 1, 2)$ は A と B の 2 つの行列の A と B に相当します。

10.18 単項演算子

- $-$ 単項マイナス演算子: 式の符号を変更します。
- $'$ はベクトルの行列またはラインビューの転置を表します。テンソル A は 2 より大きい次数であり、 A' は 2 つの第 1 指標の逆数を表示します。
- $\text{Contract}(A, i, j)$ は、 i 番目と j 番目のインデックスに対するテンソル A の収縮を表します。例えば、 $\text{Contract}(A, 1, 2)$ は、 A の $\text{Trace}(A)$ に相当します。
- $\text{Swap_indices}(A, i, j)$ はインデックス番号 i と j を交換します。たとえば、 $\text{Swap_indices}(A, 1, 2)$ は行列 A の A' に相当します。
- $\text{Index_move_last}(A, i)$ は、インデックス番号 i を最後に移動させます。例えば、 A が 4 次テンソルであるとすれば、 $\text{Index_move_last}(A, 2)$ の結果はテンソル $B_{i_1 i_3 i_4 i_2} = A_{i_1 i_2 i_3 i_4}$ になります。行列の場合、 $\text{Index_move_last}(A, 1)$ は A' に相当します。

インデックス番号 i と j を交換します。たとえば、 $\text{Swap_indices}(A, 1, 2)$ は行列 A の A' に相当します。

10.19 括弧

括弧は、標準的な方法で操作順序を変更するために使用できます。括弧が表示されていない場合、 $*$, $/$, $:$, $..$, $*$, $./$, $@$ の通常優先順位が使用されます。微分なしで、優先度が高いものは単項演算子 $-$ と $'$ に予約されています。

10.20 陽なベクトル

弱形式言語は、表記 $[a, b, c, d, e]$ 、すなわちコンマで区切られた任意の数の成分を持つ陽なベクトル（すなわち、1 テンソル）を定義することを可能にするセミコロン $[a; b; c; d; e]$ も許されます)、ベクトル全体は右括弧で始まり左括弧で終わります。成分は、いくつかの数値定数、いくつかの有効な式、試行関数を含むことができます。後者の場合、ベクトルは試行関数に関して均質でなければならない。これは、型 $[\text{Test}_u;$

Test_v] は許されません。有効な例として、u をスカラーフィールド変数とすると [5*Grad_Test_u(2), 2*Grad_Test_u(1)] です。また、オペレータ (転置) を使うと、式 [a,b,c,d,e]' は行ベクトル、つまり 1x5 行列を表します。

10.21 陽な行列

陽なベクトルと同様に、[[a,b],[c,d]] という表記で陽な行列 (すなわち、2 次テンソル) を定義することができます。つまり、カンマで区切られた任意の数の列ベクトル (セミコロンで区切られた行の構文 [a, c; b, d] も許可されます)。例えば、[[11,21],[12,22],[13,23]] と [[11,12,13;21,22,23]] はどちらも同じ 2x3 行列を表します。成分は、いくつかの数値定数、いくつかの有効な式、試行関数を含むことができます。

10.22 陽なテンソル

ネストされた形式では、任意の順序の陽なテンソルが許可されます。テンソルの次数「n」は、等次元のテンソル「n-1」の連続として書かれ、コンマで区切られています。たとえば、[[[[1,2,3],[1,2,3]],[[1,2,3],[1,2,3]]],[[[1,2,3],[1,2,3]],[[1,2,3],[1,2,3]]]] は 4 次のテンソルです。もう一つの形式は、構文 Reshape([1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,3,2,2,2]) であり、成分は Fortran の順序で指定する必要があります。

10.23 テンソル成分へのアクセス

ベクトル/行列/テンソルの成分へのアクセスは、左括弧、成分のリスト、右括弧の項をたどることによって行うことができます。たとえば、[1,1,2](3) は正しく、期待通りに 2 を返します。インデックスは 1 で始めると仮定されていることに注意してください (C++ や Python インタフェースでも)。[1,1; 2,3](2,2) と Grad_u(2,2) という表現は、u がベクトル値の宣言された変数であれば正しいです。構成要素は、一定の計算の結果であり得ることに留意してください。例えば、[1,1;2,3](1+1,a) は a が宣言された定数ですが、変数として宣言されていない場合は正しいです。コロンは Matlab のようなシンタックスのインデックスの値を置き換えて、例えば行列の行や列にアクセスすることができます。[1,1; 2,3](1,:) は行列 [1,1;2,3] の最初の行を示します。また、4 次のテンソルで使用することもできます。

10.24 定数式

- 標準表記の浮動小数点数 (例えば、3、1.456、1E-6)
- pi: 定数 Pi。

- `meshdim`: 現在のメッシュの次元 (すなわち、幾何学的節点のサイズ)
- `timestep`: この構築文字列が評価される `model` のメイン時間ステップ (`model.set_time_step(dt)` で定義されます)。純粋な作業領域では実行しないでください。
- `Id(n)`: $n \times n$ サイズの単位行列。 n は整数式でなければなりません。例えば、 `Id(meshdim)` が許可されます。
- `qdim(u)`: 変数 u の全次元 (すなわち、固定サイズ変数のサイズと FEM 変数のベクトル/テンソルフィールドの合計サイズ)
- `qdims(u)`: 変数 u のサイズ (すなわち、固定サイズ変数のサイズと FEM 変数のベクトル/テンソルフィールドの次元のベクトル)

10.25 現在の位置にリンクされた特殊な表現

- `X(i)` は i 番目の成分であり、 X は実要素上の現在の座標 (すなわち、式が評価される現在の Gauss 点のメッシュ上の位置) です。例えば、 `sin(X(1)+X(2))` は 2 次元以上のメッシュ上で有効な式です。
- `Normal` は、境界の積分を行う際の境界の外向き単位法線ベクトルです。
- `element_size` は、 (`getfem::convex_radius_estimate` を使って) 現在の要素の直径の推定値を返します。
- `element_K` は、参照 (親) 要素からの幾何学変換の勾配 (`dp-transgeo` を参照) を与えます。メッシュに混合次元の要素が含まれていない場合にのみ使用できます。
- `element_B` は、参照 (親) 要素からの幾何学変換の勾配の擬似逆行列の転置を与えます (`dp-transgeo` を参照)。メッシュに混合次元の要素が含まれていない場合にのみ使用できます。

10.26 プリントコマンド

デバッグ目的のために、 `Print(a)` コマンドはテンソル a を出力し、それをそのまま渡します。たとえば、 `Grad_u.Print(Grad_Test_u)` は `Grad_u.Grad_Test_u` と同じ効果を持ちますが、各要素の各 Gauss 積分点に対してテンソル `Grad_Test_u` を出力します。定数項は、構築の最初に 1 回だけプリントされることに注意してください。項自体の代わりに項の導関数をプリントするように表現できることにも留意してください。

10.27 テンソルを変形する

コマンド `Reshape(t, i, j, ...)` はテンソル t (式でもよい) を再形成します。唯一の制約は、成分の数に互換性が必要なことです。例えば、 `Reshape(Grad_u, 1, meshdim)` はスカラー変数の `Grad_u'` に相

当します。成分の順序は変更されず、Blas/Lapack との互換性のために Fortran の順序で古典的に格納されることに注意してください。

10.28 Trace, Deviator, Sym と Skew 演算子

Trace, Deviator, Sym および Skew 演算子は、正方行列に作用する線形演算子です。

- Trace (m) は正方行列 m の Trace (対角成分の合計) を返します。
- Deviator (m) は正方行列 m の偏差を与えます。これは $m - \text{Trace}(m) * \text{Id}(m_dim) / m_dim$ に相当します。ここで m_dim は m の次元です。
- Sym (m) は正方行列 m の対称部分、すなわち $(m + m') / 2$ を与えます。
- Skew (m) は、正方行列の Skew-対称部分、 $(m - m') / 2$ を返します。

4 つの演算子を試行関数に適用することができます。これは、例えば Grad_u が正方行列 (u と同じ次元のベクトルフィールド) であるとき、Trace (Grad_u) と Trace (Grad_Test_u) は有効であることを意味します。(すなわち、u と次元として同じ次元のベクトルフィールドです。)

10.29 非線形演算子

弱形式言語は、あらかじめ定義された非線形演算子を提供します。各非線形演算子は、その第 1 および第 2 導関数と共に利用可能です。非線形演算子は、式にいくつかの試行関数が含まれていない限り、適用できます。

- ベクトルまたは行列である v に対して Norm (v) を実行すると、ベクトルのユークリッドのノルムまたは Frobenius 行列のノルムが与えられます。
- ベクトルまたは行列のユークリッドのノルムの二乗は、Frobenius 行列のノルムベクトルの場合、これは $v \cdot v$ に相当し、行列は $m:m$ に相当します。
- v に対して Normalized (v) は v をユークリッド (ベクトルの場合) または Frobenius (行列の場合) ノルムで割られる。v が 0 に近いときの問題を避けるため、`Normalized_reg (v, 1E-25)` として実装されています。注意して使用してください。Normalized (v) * Norm (v) 起源の導関数は間違っています (それは消滅します)、v の導関数とは非常に異なると考えてください。
- v の Normalized_reg (v, eps) ベクトルまたは行列は `Normalized (v)` : $v / \sqrt{(|v| * |v| + eps * eps)}$ の正規化されたバージョンを与えます。
- ベクトルまたは行列 v とスカラーの r の Ball_projection (v, r) は、原点を中心にした半径 r のボールに v の投影を与えます。

- `Det (m)` は正方行列 m の行列式を与えます。
- `Inv (m)` は正方行列 m の逆行列を返します。2 次導関数は、6 次テンソルであるため利用できません。つまり、ポテンシャルエネルギーの記述には `Inv (m)` は使用できません。
- `Expn (m)` は正方行列 m の指数を与えます。
- `Logm (m)` は正方行列 “ m ” の対数を返します。
- `Matrix_I2 (m)` は $(\text{sqr}(\text{Trace}(m)) - \text{Trace}(m*m))/2$ で定義される正方行列 m の第 2 の不変量を返します。
- `Matrix_J1 (m)` は、 $\text{Trace}(m) \text{pow}(\text{Det}(m), -1/3)$ で定義された正方行列の最初の不変量を修正したものです。
- `Matrix_J2 (m)` は $\text{Matrix_I2}(m) * \text{pow}(\text{Det}(m), -2/3)$ によって定義された正方行列の修正された最初の不変量を返します。

10.30 マクロ定義

弱形式言語を使用すると、`model` または `ga_workspace` オブジェクトであらかじめ定義されているか、構築文字列の先頭に直接定義されているマクロを使用できます。`ga_workspace` または `model` オブジェクトへの定義は次のように行われます。

```
workspace.add_macro(name, expr)
```

または:

```
model.add_macro(name, expr)
```

マクロの構築文字列への定義は、正規表現の前に次の構文でセミコロンで区切られて挿入されます。

```
"Def name:=expr; regular_expression"
```

`name` はマクロ名であり、弱形式言語で使用され、マクロのパラメータも含みます。`expr` は弱形式言語の有効な式です（それ自体がいくつかのマクロ定義を含むかもしれません）。たとえば、パラメータを持たない有効なマクロは次の通りです。

```
model.add_macro("my_transformation", "[cos(alpha)*X(1);sin(alpha)*X(2)]");
```

`alpha` は有効な宣言された変数またはデータでなければなりません。2 つのパラメータを持つ有効なマクロは、例えば次の通りです。

```
model.add_macro("ps(a,b)", "a.b");
```

次の構築文字列は有効です (u が有効な変数である場合)。

```
"Def ps(a,b):=a.b; ps(Grad_u, Grad_Test_u) "
```

パラメータは Grad_、Hess_、Test_、Test2_ の接頭辞の後に固定することができ、次の構築文字列が有効です。

```
"Def psgrad(a,b):=Grad_a.Grad_b; psgrad(u, Test_u) "
```

または 2 つのマクロの組み込みです。

```
"Def ps(a,b):=a.b; Def psgrad(a,b):=ps(Grad_a,Grad_b); psgrad(u, Test_u) "
```

マクロは、次のように `ga_workspace` オブジェクトまたは `model` オブジェクトから削除できます。

```
workspace.del_macro(name)
model.del_macro(name)
```

構築文字列の先頭に定義されたマクロは構築文字列でのみ定義され、後で `model` または `ga_workspace` オブジェクトに追加することなく使用することはできないことに注意してください。

このマクロは字句解析フェーズでインライン展開されます。コンパイル段階では、繰り返し式は自動的に分解され、1 回だけ計算されます。

10.31 陽な微分

作業領域オブジェクトは、より低い序列のものを自動的に微分します。しかし、変数に対して式を陽に微分することもできます。注目すべきは、自動微分は `model/workspace` の宣言されたすべての変数に関して微分を実行するが、これは例えばポテンシャルエネルギーを使用する場合に必ずしも期待される挙動ではないということです。構文は次の通りです。

```
Diff(expression, variable)
```

例えば、次の式となります。

```
Diff(u.u, u)
```

結果は次の通りです。

```
2*(u.Test_u)
```

そのため。

```
Grad_u: Grad_test_u + Diff(u,u, u)
```

これは有効な式です。方向を指定するために Diff コマンドに 3 番目の引数を追加することができます。

```
Diff(expression, variable, direction)
```

その場合、Test_variable を variable と同じ次元である式 direction で置き換えます。direction 方向の variable に対して expression の導関数を計算します。例えば。

```
Diff(u,u, u, v)
```

結果は次の通りです。

```
2*(u.v)
```

v が u と同じ次元の有効な式の場合、

10.32 陽な勾配

次の関数を使って式の勾配の記号計算を求めることができます。

```
Grad(expression)
```

それは可能な限り計算されます。制限として、*GetFEM++* は形状関数の 2 階微分に限定され、非線形演算子は 1 次微分と 2 次微分のみを与えられています。

もちろん。

```
Grad(u)
```

次と同等です。

```
Grad_u
```

変数 u に対して。

10.33 補間変換

Interpolate 演算は、メッシュの異なる部分に定義された量または異なるメッシュに定義された量の間の積分を計算することを可能にします。それは Mortar 行列を計算するか、または周期的な条件を考慮に入れることができる強力な演算です。しかしながら、これは無視できない計算コストを有するかもしれない補間に基づいていることを覚えておいてください。

この機能を使用するために、ユーザはまず、現在の積分ポイントと同じメッシュまたは別のメッシュ上にあるポイントとの間のマップを記述する補間変換を `workspace` または `model` オブジェクトに宣言する必要があります。

異なる種類の変換を記述することができます。いくつかの種類の変換が実装されています。以下で説明する第 1 のものは、式によって記述される変換です。2 番目はレイトレーシング接触検出に相当します（[レイトレーシング補間変換](#) を参照）。次のセクションでは、他の変換（隣接要素と要素の外挿）について説明します。

式で定義された変換は、次のコマンドにより `workspace` または `model` に追加できます。

```
add_interpolate_transformation_from_expression
  (workspace, transname, source_mesh, target_mesh, expr);
```

または:

```
add_interpolate_transformation_from_expression
  (model, transname, source_mesh, target_mesh, expr);
```

`workspace` は `workspace` オブジェクト、`model` は `model` オブジェクト、`transname` は変換に与えられた名前、`source_mesh` は積分が行われるメッシュ、`target_mesh` 補間が実行されるメッシュで `expr` は、`workspace/model` の変数への参照を含む高水準の汎用的な弱形式言語の正規表現です。

例えば、次の式は、

```
add_interpolate_transformation_from_expression
  (model, "my_transformation", my_mesh, my_mesh, "X-[1;0]");
```

現在の位置にある式を最初の座標に対して-1 のシフトで積分することができます。この単純な種類の変換を使用して、周期的な条件を規定することができます。

もちろん、もっと複雑な表現を使うかもしれません。

```
add_interpolate_transformation_from_expression
  (model, "my_transformation", my_mesh, my_second_mesh, "[X[1]cos(X[2]); X[1]sin(X[2])]");

add_interpolate_transformation_from_expression
  (model, "my_transformation", my_mesh, my_mesh, "X+u");
```

ここで `u` は作業空間/`model` のベクトル変数です。

`workspace/model` で変換が定義されると、変数または試行関数、これらの式の 1 つのために境界への位置または単位法線ベクトルを補間できます。

```
Interpolate(Normal, transname)
Interpolate(X, transname)
Interpolate(u, transname)
Interpolate(Grad_u, transname)
Interpolate(Div_u, transname)
```

```
Interpolate(Hess_u, transname)
Interpolate(Test_u, transname)
Interpolate(Grad_Test_u, transname)
Interpolate(Div_Test_u, transname)
Interpolate(Hess_Test_u, transname)
```

ここで u は補間される変数の名前です。

例えば、乗算器 λ によって、変数 u とその補間（例えば、周期的境界条件を規定するため）との等価を規定する構築式は、次のように書くことができます。

```
(Interpolate(u, my_transformation) - u) * lambda
```

(`interface/tests/matlab` 内の `demo_periodic_laplacian.m` を参照してください。)

場合によっては、変換された点がターゲットメッシュの外側にある場合、点の補間が失敗することがあります。両方とも、このケースを処理し、変換が他のいくつかの場合を微分できるようにするために（両方とも、剛体と `Raytracing_interpolate_transformation` の変形可能なものとの間の差異については、[レイトレーシング補間変換](#) を参照）、変換は整数識別子が弱形式言語に返されます。この識別子の値 0 は、ターゲットメッシュ上の対応する位置が見つからなかったことを意味します。1 の値は、対応する点が見つかったことを意味します。この識別子は、弱形式言語の次の特別なコマンドにより使用できます。

```
Interpolate_filter(transname, expr, i)
```

`transname` は変換の名前、`expr` は評価される式、`i` は式が計算されるべきかを返す整数識別子です。`i` は省略することができ、この場合、式は非ゼロの識別情報（すなわち、対応する点が見つかったとき）について評価されることに留意してください。例えば、変数 “ u ” とその補間との等価を規定する以前の構築式は、次のように書くことができます。

```
Interpolate_filter(transname, Interpolate(u, my_transformation) - u) * lambda
+ Interpolate_filter(transname, lambda * lambda, 0)
```

その場合、等式は、変換が成功する領域の部分でのみ規定され、他の部分では、乗数は消滅するように強制されます。

注意：変換でいくつかの変数を使用される場合、接触 `model` の計算でこれらの依存性が考慮されていると考える必要があります。しかし、使用される変数に関する変換の 2 次導関数は実装されていません。従って、このような変換は、2 回微分することができないので、電位の定義には使用できません。

10.34 要素外挿変換

別の要素（一般的に隣接要素）に関してある量の外挿による評価を可能にするために、特定の変換（前の節を参照）が定義されます。点の位置は変わらないので、厳密には変換ではありませんが、評価は試行関数を外挿する別

の要素に対して行われます。この変換は、現実の領域との十分に大きな交差を有する隣接要素のいくつかの量を、現在の要素が小さい場合にそれらを評価するよりも外挿することがより確実である仮想領域法（切断要素を有する）において、このような実領域との交差変換を `model` または `workspace` に追加するには次の関数を使用します。

```
add_element_extrapolation_transformation
(model, transname, my_mesh, std::map<size_type, size_type> &elt_corr);
```

```
add_element_extrapolation_transformation
(workspace, transname, my_mesh, std::map<size_type, size_type> &elt_corr);
```

マップ `elt_corr` には、変換が適用される要素と、補外が行われる必要があります。それぞれの要素とへの対応関係が含まれている必要があります。マップにリストされていない要素では、変換は適用されず、評価は現在の要素に対して正常に実行されます。

以下の関数は、以前に追加された要素補間変換の要素対応を変更することを可能にします。

```
set_element_extrapolation_correspondance
(model, transname, std::map<size_type, size_type> &elt_corr);
```

```
set_element_extrapolation_correspondance
(workspace, transname, std::map<size_type, size_type> &elt_corr);
```

10.35 要素間エッジ/面間の不連続性の評価します。

`neighbour_elt` と呼ばれる特定の補間変換（前のセクションを参照）は、デフォルトですべての `model` で定義されています。この変換は、メッシュの内部エッジ/面、すなわち少なくとも2つの要素によって共有される要素面で計算が行われる場合にのみ使用できます。これは、要素間の面で変数の不連続ジャンプを計算することを目的としています。不連続 Galerkin と内部ペナルティ法、ゴーストペナルティ項または事後推定を実装するのに特に適しています。表現は、

```
Interpolate(Normal, neighbour_elt)
Interpolate(X, neighbour_elt)
Interpolate(u, neighbour_elt)
Interpolate(Grad_u, neighbour_elt)
Interpolate(Div_u, neighbour_elt)
Interpolate(Hess_u, neighbour_elt)
Interpolate(Test_u, neighbour_elt)
Interpolate(Grad_Test_u, neighbour_elt)
Interpolate(Div_Test_u, neighbour_elt)
Interpolate(Hess_Test_u, neighbour_elt)
```

（他の補間変換の場合と同様に）現在の点ではなく隣接要素上のフィールドを計算することができます。もちろん、`Interpolate(X, neighbour_elt)` は `X` と同じ結果を返すのでそれほど特別なものではありません。同様

に、ほとんどの場合、`Interpolate(X, neighbour_elt)` は、関心のある 3D メッシュの 2D シェル要素を除いて、`Normal` の反対側を返します。

変数 `u` のジャンプは、

```
u-Interpolate(u, neighbour_elt)
```

ジャンプのペナルティ・タームを書くことができます。

```
(u-Interpolate(u, neighbour_elt))*(Test_u-Interpolate(Test_u, neighbour_elt))
```

メッシュのすべての内部面の集合を表す領域は、次の関数により与えられます。

```
mr_internal_face = inner_faces_of_mesh(my_mesh, mr)
```

ここで `mr` はオプションのメッシュ領域です。 `mr` が指定されていれば、この領域に関して内部の面だけが返されます。重要な点は、各面が 1 回だけ表され、2 つの隣接要素の間で任意に選択されることです。

使用例は `interface/tests/python/demo_laplacian_DG.py` や `interface/tests/matlab/demo_laplacian_DG.m` を参照してください。

他の補間変換と比較すると、この変換はより最適化され、有限要素変換および幾何変換前処理で恩恵があります。

10.36 ダブル領域の積分または項（畳み込み - カーネル - 交換積分）

いくつかの非常に特殊な場合には、2 つの領域の直積、すなわち例えば 2 重積分の積分を計算することに着目する必要があります。

$$\int_{\Omega_1} \int_{\Omega_2} k(x, y) u(x) v(y) dy dx,$$

ここで $k(x, y)$ は与えられたカーネルです、 u は Ω_1 上で定義された量で v は Ω_2 上で定義された量です、 Ω_1 と Ω_2 は同じ領域にあります。これは、そのような積分を計算するか、または 2 つの異なるドメイン上に定義された 2 つの変数間の相互作用項を定義する際に着目する必要があります。

注意: 当然のことながら、この種の項は、通常に完全に計算された剛性または接線行列に関連するため、細心の注意を払って使用する必要があります。

`GetFEM++` の弱形式言語はそのような項を計算するためのメカニズムを与えます。まず、第 2 領域は、その積分法を使用して `workspace/model` 内で宣言されなければなりません。標準の第 2 領域の追加は、次の 2 つの関数のうちの 1 つを使用して行うことができます。

```
add_standard_secondary_domain(model, domain_name, mim, region);
```

```
add_standard_secondary_domain(workspace, domain_name, mim, region);
```

model または workspace は第 2 領域を宣言しなければならない model または workspace で、domain_name はメッシュ領域と積分法とともにこの領域を識別する文字列です。積分法を mim、メッシュ領域を region と呼びます。これらの標準的な第 2 領域では、第 1 領域の各要素について領域全体で積分が行われることに着目してください。第 1 領域の要素に関して必要な要素への積分を制限する特定の第 2 領域を実装することに着目してください。GetFEM++ 中の構造体はこれ専用です。

第 2 領域が宣言されると、現在の領域と第 2 領域の直接的な生成物に弱形式言語をアセンブルし、add_expression ``メソッド

```
workspace.add_expression(expr, mim, region, derivative_order, secondary_domain)
add_twodomain_source_term(model, mim, expr, region, secondary_domain)
add_linear_twodomain_term(model, mim, expr, region, secondary_domain)
add_nonlinear_twodomain_term(model, mim, expr, region, secondary_domain)
```

Python / Scilab / Matlab インタフェースを利用するには、“gf_asm”コマンドと“model”オブジェクトに関するドキュメントを参照してください。

弱形式言語の表現中で、式により、境界への単位法線ベクトル、変数の現在の位置または値を参照することができます。

```
Secondary_domain (Normal)
Secondary_domain (X)
Secondary_domain (u)
Secondary_domain (Grad_u)
Secondary_domain (Div_u)
Secondary_domain (Hess_u)
Secondary_domain (Test_u)
Secondary_domain (Grad_Test_u)
Secondary_domain (Div_Test_u)
Secondary_domain (Hess_Test_u)
```

たとえば、

$$\int_{\Omega_1} \int_{\Omega_1} e^{-\|x-y\|} u(x)u(y)dydx,$$

言語表現のために次の弱点に対応します。

```
exp(Norm(X-Secondary_domain(X)))*u*Secondary_domain(u)
```

10.37 初等変換

基本変換は、要素水準で局所自由度に適用される要素に依存する可能性のある行列によって与えられる形状関数の線形変換です。基本変換の定義の例はファイル src/getfem_linearized_plates.cc にあります。これは、例えば、MITC 要素で使用されるような縮小を実行するために、より低い水準の要素上の有限要素の局所射影を定義することを目的とします。

変換が定義されると、`model/workspace` に次のメソッドを追加することができます。

```
model.add_elementary_transformation(transname, pelementary_transformation)
```

ここで `pelementary_transformation` は `virtual_elementary_transformation` から派生したオブジェクトへのポインタです。`model/workspace` に追加されると、弱形式言語で次の式を使用することができます。

```
Elementary_transformation(u, transname)
Elementary_transformation(Grad_u, transname)
Elementary_transformation(Div_u, transname)
Elementary_transformation(Hess_u, transname)
Elementary_transformation(Test_u, transname)
Elementary_transformation(Grad_Test_u, transname)
Elementary_transformation(Div_Test_u, transname)
Elementary_transformation(Hess_Test_u, transname)
```

ここで `u` は `model/workspace` の FEM 変数の 1 つです。現時点では、利用可能な基本変換は、(`src/getfem/getfem_linearized_plates.h` で定義されている) 次の関数により追加できる 2 次元要素の回転 RT0 要素の投影です。

```
add_2D_rotated_RT0_projection(model, transname)
```

10.38 Xfem 不連続性評価 (`mesh_fem_level_set` を使用)

GetFEM++ 5.1. について。レベル集合 (`fem_level_set` または `mesh_fem_level_set` オブジェクトを使用) によって有限要素法を切断する場合、変数の不連続ジャンプ、または勾配または平均値のジャンプを組み込むのが面白いことがよくあります。このために、弱形式言語は、`u` の FEM 変数を定義しています。

```
Xfem_plus(u)
Xfem_plus(Grad_u)
Xfem_plus(Div_u)
Xfem_plus(Hess_u)
Xfem_plus(Test_u)
Xfem_plus(Test_Grad_u)
Xfem_plus(Test_Div_u)
Xfem_plus(Test_Hess_u)

Xfem_minus(u)
Xfem_minus(Grad_u)
Xfem_minus(Div_u)
Xfem_minus(Hess_u)
Xfem_minus(Test_u)
Xfem_minus(Test_Grad_u)
```

```
Xfem_minus(Test_Div_u)
Xfem_minus(Test_Hess_u)
```

これは、連続性の 2 つのゾーンを分離する曲線/表面、すなわち、考慮されるレベル集合関数のゼロレベル集合 (mesh_im_level_set オブジェクトを使用する) で評価 (積分) が行われる場合にのみ利用可能です。例えば、u 変数のジャンプは次のようになります。

```
Xfem_plus(u) - Xfem_minus(u)
```

平均値は次のようになります。

```
(Xfem_plus(u) + Xfem_minus(u)) / 2
```

The value ``Xfem_plus(u)`` is the value of ``u`` on the side where the corresponding level-set function is positive.

Additionally, note that, when integrating on a level-set with a ``mesh_im_level_set`` object, ``Normal`` is the normal vector pointing outwards from the level-set.

10.39 構築中の getfem::im_data オブジェクトへのサブ式の格納

getfem::im_data オブジェクトに応じてベクトルに格納することができます。たとえば、この計算を別の構築で使用するために、構築計算の一部にすることができます。これは、同じ式を 2 回計算しないようにする補間関数の代替方法です。

このような割り当てを構築に追加する方法は、model または ga_workspace です。

```
model.add_assembly_assignments(dataname, expr, region = size_type(-1),
                               order = 1, before = false);

workspace.add_assignment_expression(dataname, expr,
                                    region = mesh_region::all_convexes(), order = 1, before = false)
```

組み立て時に評価される式 *expr* を追加し、im_data 型でなければならないデータ *dataname* に代入します。order は、この割り当てが行われなければならない構築の次数を表します (ポテンシャル (0)、弱形式 (1) または接線のシステム (2) または次数ごと (-1))。before = 1 の場合、代入は他の構築項の計算の前に実行されるため、データを構築の残りの部分で中間結果として使用することができます (データとしてはまだ考慮していることに注意してください、接線系の式の導出は行われません)。before = 0 (デフォルト) の場合、構築の項の後に割り当てが行われます。

さらに、model では、次のメソッドがあります。

```
model.clear_assembly_assignments()
```

以前に追加された構築割り当てをすべてキャンセルすることができます。

第 11 章

任意の項を計算する - 低レベルの汎用的な構築手順

このセクションでは、*GetFEM++* で実装された汎用構築プロシージャの最初のバージョンを説明します。これにより、線形の場合に任意の行列の集合を容易に作成可能になります。非線形の場合、特殊な “non_linear_term” オブジェクトを実装する必要があります。これはややこしいかもしれませんが、*GetFEM++* という非常に低レベルの内部ツールを使用する必要があります。高レベル汎用構築は、この困難を回避するために開発されました（任意の項を計算する - 高水準の汎用的な構築手順 - 弱形式言語 参照）。

ファイル `getfem/getfem_assembling.h` で見る事ができるように、以前の構築プロシージャはすべて `getfem::generic_assembly` オブジェクトを使用しどのようにしなければならないかについて示しています。例えば、スカラー FEM のためのボリュームソース項の構築は、コードの以下の抜粋を用いて行われます。

```
getfem::generic_assembly assem;
assem.push_im(mim);
assem.push_mf(mf);
assem.push_mf(mfdata);
assem.push_data(F);
assem.push_vec(B);
assem.set("Z=data(#2);"
          "V(#1)+=comp(Base(#1).Base(#2))(:,j).Z(j);");
assem.assembly();
```

最初の命令はオブジェクトを宣言し、使用するデータを設定します積分メソッドを保持する *mesh_im* オブジェクト、2つの *mesh_fem* オブジェクト、入力データ *F*、および宛先ベクトル *B* を含みます。

入力データは *mfdata* で定義されているベクトル *F* です。1つは $\sum_j f_j (\int_{\Omega} \phi^i \psi^j)$ を評価することです。命令はメッシュの凸状の *cv* ごとに実行されます。項 #1 と項 #2 は最初の *mesh_fem* 2つ目は *mf* と *mfdata* です。`Z=data(#2);` はそれぞれの凸包について、`push_data` で与えられた最初のデータ引数の値を、`push_data` に対応するインデックスで受け取ることを示します。2番目の (#2) の凸包に付けられた自由度 *mesh_fem* (こ

ここでは $Z = F[\text{mfdata.ind_dof_of_element}(cv)]$ となります。

$V(\#1) += \dots$ の部分は、次の式の結果が出力ベクトルに蓄積されることを意味します (`push_vec` で提供されます)。ここでもまた、 $\#1$ は、最初の ($\#1$) `mesh_fem` に関して、現在の凸の自由度に対応するインデックスに結果を書き込むことを意味します。

右側の `comp(Base(#1).Base(#2))(:,j).Z(j)` には 2 つの操作が含まれています。最初のもは、凸包のテンソルの計算です: `comp(Base(#1).Base(#2))` は 2 次元のテンソルとして評価されます $\int \phi^i \psi^j$ をすべての自由度について計算します `mf` と現在の凸面に付けられた `mfdata` の j 。次の部分はリダクション演算で、`C(:,j).Z(j)`: それぞれの指数 (ここでは j) を合計します。結果は $\sum_j c_{i,j} z_j$ となります。

`comp(Base(#1).Base(#2))` の内部で使用される積分法は `mim` から取得されます。別の `mesh_im` オブジェクトから積分メソッドを使用する必要がある場合は、例えば、`comp(\%2, Base(#1).Grad(#2))` のように `comp` の最初の引数として指定することができます。オブジェクト (`getfem++-2.0` の新機能)。

他の例として、ベクトル Laplacian の剛性行列を構築する

```
getfem::generic_assembly assem;
assem.push_im(mim);
assem.push_mf(mf);
assem.push_mf(mfdata);
assem.push_data(A);
assem.push_mat(SM);
assem.set("a=data$1(#2);"
          "M$1(#1,#1)+=sym(comp(vGrad(#1).vGrad(#1).Base(#2))(:,j,k,(:,j,k,p)).a(p))");
assem.assembly();
```

出力は、`assem.push_mat(SM)` で挿入された疎行列に書き出されます。M\$1(#1,#1) の \$1 は最初の行列を “push” していることを示しています (オプションですが、2 つの行列を構築するならば、2 つ目はこのように参照する必要があります)。`sym` 関数は、結果が対称であることを保証します (これが行われないと、いくつかの丸め誤差が対称性を取り消し、構築が少し遅くなることに留意してください)。次に、`comp` 部分は 7 次テンソルを評価し、

$$\int \partial_k \varphi_j^i \partial_n \varphi_m^l \psi^p,$$

ここで、 φ_j^i は `mf` の i 次の基底関数の j 次成分で、 ψ^p は (スカラー) 2 番目の `mesh_fem` の基底関数を構築したので

$$\int a(x) \cdot \nabla \phi^i \cdot \nabla \phi^j, \quad \text{with} \quad a(x) = \sum_p a^p \psi^p(x),$$

換算は次のとおりです。

$$\sum_{j,k,p} \left(\int \partial_k \varphi_j^i \partial_k \varphi_j^m \right) a^p$$

comp 関数で Grad の代わりに vGrad が使われました。なぜなら、私たちは *vector Laplacian* をアセンブルしているためです。そのため、それぞれの vGrad 部分は 3 次元自由度番号、成分番号、および派生番号)。スカラーラプラシアンの場合、`comp(Grad(#1).Grad(#1).Base(#2))(:,k, :,k,p).a(p)` を使うことができます。しかし、ベクトル形式はベクトルとスカラーの両方の場合に機能する利点があります。

最後の命令である `assem.assembly()` は各凸包の式を評価します。境界を越えて構築を呼び出すには、`assem.assembly(rg)` を呼び出します。ここで `rg` は `getfem::mesh_region` オブジェクト。 `rg` も数値であるかもしれません。その場合メッシュ領域は `mim.linked_mesh().region(rg)` です。

3 番目の例は、メッシュ境界上のスカラーまたはベクトルフィールドの L^2 ノルムを計算する方法を示します:

```
assem.push_im(mim);
assem.push_mf(mf);
assem.push_data(U);
std::vector<scalar_type> v(1);
assem.push_vec(v);
assem.set("u=data(#1);"
         "V() += u(i) . u(j) . comp(vBase(#1) . vBase(#1)) (i, k, j, k)");
assem.assembly(boundary_number);
```

これは読みやすいです。 `assembly` が返ってくる時、 `v[0]` は

$$\sum_{i,j,k} \left(\int_{boundary} u_i \varphi_k^i u_j \varphi_k^j \right)$$

4 番目と最後の例は、完全な Hooke テンソルによる線形弾性問題の (準最適な) 構築を示しています:

```
assem.set("h=data$1(qdim(#1), qdim(#1), qdim(#1), qdim(#1), #2);"
         "t=comp(vGrad(#1) . vGrad(#1) . Base(#2));"
         "e=(t(:,2,3, :,5,6, :)+t(:,3,2, :,5,6, :)+t(:,2,3, :,6,5, :)+t(:,3,2, :,6,5, :))/4;"
         "M(#1, #1) += sym(e(:,j,k, :,m,n,p) . h(j,k,m,n,p))");
```

元の方程式は次のとおりです。

$$\int \varepsilon(\varphi^i) : \sigma(\phi^j), \quad \text{with} \quad \sigma(u)_{ij} = \sum_{kl} h_{ijkl}(x) \varepsilon_{kl}(u)$$

h は Hooke テンソル、そして $:$ は行列間のスカラー積を意味します。定数ではないと仮定しているため、第 2 の `mesh_fem`: $h_{ijkl}(x) = \sum_p h_{ijkl}^p \psi^p$ に h が与えられます。したがって、最初の行は、最初のデータが実際に 5 次元のテンソルであることを宣言します。最初の 4 番目のものはすべて最初の `mesh_fem` のターゲット次元に等しく、最後のものは次の次元数に等しくなります。第 2 の自由 `mesh_fem`。 `comp` 部分はベクトルラプラシアンの場合と同じ 7 次テンソルを計算します。このテンソルから、順列を使って $\varepsilon(\varphi^i)_{jk} \varepsilon(\phi^l)_{mn} \psi^p$ を評価し最後に表現はフックテンソルに換算されます。

11.1 comp コマンドの中で利用可能な操作

- `Base(#i)`: i 次 `mesh_fem` の基底関数の値を評価します。
- `Grad(#i)`: i 次 `mesh_fem` の基底関数の勾配の値を評価します。
- `Hess(#i)`: i 次 `mesh_fem` の基底関数のヘッセ行列の値を評価します。
- `Normal()`: 単位法線を評価します (ボリウム積分には使用しないでください)。
- `NonLin$x(#mf1, ... #mfn)`: `push_nonlinear_term(pnonlinear_elem_term)` で挿入された、リストされた `mesh_fem` オブジェクトを使用した x 次非線形項の計算。
- `GradGT()`、`GradGTInv()`: 現在の凸包の幾何学的変換の勾配 (およびその逆数) を評価します。

ノート: `comp` コマンドの中の任意のデータオブジェクトを参照し、`comp()` の中で縮小を実行することができます。この機能は非線形項の組み立てを高速化するために有用です (ファイル `getfem/getfem_nonlinear_elasticity.h` の使用例を参照)。

11.2 他の操作

スライスと縮小演算と混在することがあります。 `t(:, 4, i, i)` は第 2 次元のインデックス 4 でスライスを取り、次元 3 と 4 の対角を縮小します。スライスのインデックス番号は 1 ではなく 0 で始まる!!

`mdim(#2)` は第 2 の `mesh_fem` に関連するメッシュ次元として評価され、`qdim(#2)` は `mesh_fem` の対象次元です。

テンソルの対角は `t{:, :, 3, 3}` で得ることができます。(厳密には `t{1, 2, 3, 3}` と同等です: コロンは読みやすさを向上させるためだけです)。これは、並べ替え操作と同じ演算子です。 `t{:, :, 1, 1}` や `t{:, :, 4, 4}` は有効な操作ではないことに注意してください。

`print` コマンドは、テンソルを見るために使うことができます: `"print comp(Base(#1));"` は各凸包の基底関数の積分を出力します。

2 つ以上のデータ配列、出力配列または出力疎行列がある場合、`data$2`, `data$3`, `V$2`, `M$2` などを使用できます。

第 12 章

いくつかの標準構築手順（低レベル汎用構築）

ファイル `getfem/getfem_assembling.h` で定義されている手続きは、古典的な偏微分方程式の問題のいくつかの剛性行列、質量行列、境界条件の構築を可能にします。任意の行列ライブラリで使用するために、すべてのプロシージャにはベクトルと行列のテンプレートパラメータがあります。

注意：構築手順では、同じ行列/ベクトルに対して複数の構築操作を実行する可能性があるため、構築の最初に行列/ベクトルを初期化しません。したがって、最初の構築操作の前にマトリックス/ベクトルを初期化する必要があります。

12.1 Laplacian (Poisson) 問題

この問題を解決するための組み立て手順が定義されています。

$$\begin{aligned} -\operatorname{div}(a(x) \cdot \operatorname{grad}(u(x))) &= f(x) \text{ in } \Omega, \\ u(x) &= U(x) \text{ on } \Gamma_D, \\ \frac{\partial u}{\partial \eta}(x) &= F(x) \text{ on } \Gamma_N, \end{aligned}$$

ここで、 Ω は任意の次元のオープン領域です Γ_D と Γ_N は、 $a(x)$ は与えられた係数であり、 $f(x)$ は与えられたソースの項である $U(x)$ は次の式 Γ_N 上の $u(x)$ の正規方程式です。剛性行列をアセンブルするために呼び出される関数は:

```
getfem::asm_stiffness_matrix_for_laplacian(SM, mim, mfu, mfd, A);
```

ここで

- `SM` は正しい次元（つまり、`mfu.nb_dof()`）を持つ任意の型の行列です。
- `mim` は使用される積分方法を定義する型 `getfem::mesh_im` の変数です。
- `mfu` は解の有限要素法を定義する型 `getfem::mesh_fem` の変数です。

- `mfd` は係数 $a(x)$ が定義されている有限要素法を記述する型 `getfem::mesh_fem` (おそらく `mfu` に等しい) の変数です。
- A は `mfd` の各自由度に対するこの係数の値の (実数または複素数) ベクトルです。

両方の `mesh_fem` は同じメッシュ (つまり、`&mfu.linked_mesh() == &mfd.linked_mesh()`) を使用する必要があります。

基本行列を計算するのに使われる `mim` に格納されている積分方法は十分な次元のものを選択することが重要です。次数は `mfu` の要素の多項式の次数、`mfd` 非線形の場合の幾何変換を考慮して決定する必要があります。例えば、線形幾何学変換の場合、`mfu` は P_K 有限要素法、`mfd` は P_L 有限要素法です。積分次数は次数 $\geq 2(K-1) + L$ で選択する必要があります、それゆえ `SM` の構築の間に計算される要素積分は $\int \nabla \varphi_i \nabla \varphi_j \psi_k$ のようになります (それと φ_i は `mfu` の基底関数で、 ψ_i は `mfd` の基底関数です)。

ソース項を構築するのに呼び出す関数は:

```
getfem::asm_source_term(B, mim, mfu, mfd, V);
```

B は正しい次元 (まだ `mfu.nb_dof()`) を持つ型のベクトルで、`mim` は使用される積分方法を定義する `getfem::mesh_im` 型の変数です、`mfd` はそれぞれの自由度の $f(x)$ 上の有限要素法が記述されている型 `getfem::mesh_fem` (おそらく `mfu` に等しい) の変数です、そして V は `mfd` 上の各自由度の $f(x)$ の値のベクトルです。

関数 `asm_source_term` にはオプションの引数があります。これは `getfem::mesh_region` (または単に整数 i 、`mim.linked_mesh().region(i)` が考慮されます)。したがって、Neumann 条件 Γ_N と同等の関数は:

```
getfem::asm_source_term(B, mim, mfu, mfd, V, nbound);
```

`mim`、`mfu`、`mfd` のリンクメッシュでは、`nbound` は境界 Γ_N のインデックスです。

線形システムを変更する、または明示的に Dirichlet 条件の核に還元することにより、 Γ_D の Dirichlet 条件を考慮したり、線形システムを変更したり、Dirichlet 条件のカーネルに明示的に縮小するには、2つの方法があります (実際には、Lagrange 乗数を使用するか、ペナルティを使用することも可能です)。最初の方法として、以下の関数が定義されています:

```
getfem::assembling_Dirichlet_condition(SM, B, mfu, nbound, R);
```

ここで、`nbound` は境界のインデックスです Γ_D Dirichlet 条件が適用される場合、 R は `mfu` の各自由度の $R(x)$ の値のベクトルです。この操作は、剛性行列 `SM` を変換するので、最後の操作でなければなりません。これはラグランジュ要素に対してのみ機能します。最後に、離散システムを得るには:

$$[SM]U = B,$$

ここで、 U は離散未知数です。

2 番目の方法では、もっと汎用的な:

```
getfem::asm_dirichlet_constraints(H, R, mim, mf_u, mf_mult,
                                mf_r, r, nbound).
```

解 u を満たす汎用的な線形制約 **Dirichlet** 条件を参照してください。この関数は、`mf_mult` で定義された乗数空間の全ての v に対して $\int_{\Gamma} u(x)v(x) = \int_{\Gamma} r(x)v(x)$ の型の **Dirichlet** 条件の構築を行います。`mf_u` があまりにも“複雑”な場合を除いて、有限要素法の `mf_mult` は `mf_u` と同じように選ばれることがよくあります。

この関数はこれらの制約を新しい線形システム $Hu = R$ に組み立てるだけです。また、“単純な”制約行列を得るためにいくつかの追加の単純化を行います。

それでは、

```
ncols = getfem::Dirichlet_nullspace(H, N, R, Ud);
```

Dirichlet 条件を満たすベクトル U_d と、 H のカーネルの直交基底 N を返す。したがって、解く必要のある離散システムは、

$$(N'[SM]N)U_{int} = N'(B - [SM]U_d),$$

解は $U = NU_{int} + U_d$ です。出力行列 N は $nbdof \times nbdof$ (疎行列) 行列でなければなりません、`ncols` 列にリサイズする必要があります。出力ベクトル U_d は、 $nbdof$ ベクトルでなければなりません。このアプローチの大きな利点は汎用性であり、有限要素法の `mf_u` が **Lagrange** 型であることは規定されていません。`mf_u` と `mf_d` が異なる場合、有限要素 `mf_u` 上のデータの投影 (L^2 ノルムに関して) が暗黙的にあります。

より汎用的なスカラー楕円方程式 $\text{div}(A(x)\nabla u)$ を扱いたい場合、次の式を使用する必要があります。ここで、 $A(x)$ は正方行列です。

```
getfem::asm_stiffness_matrix_for_scalar_elliptic(M, mim, mfu,
                                                mfd, A);
```

行列データ A は、`mfd` 上で定義し $n \times n \times nbdof$ テンソル (**Fortran** 順) を表すベクトルである必要があります。 n は `mfu` のメッシュの次元であり、 $nbdof$ は `mfd` の自由度数です。

12.2 線形弾性問題

以下の関数は、線形弾性のための剛性行列を組み立てます:

```
getfem::asm_stiffness_matrix_for_linear_elasticity(SM, mim, mfu,
                                                  mfd, LAMBDA, MU);
```

SM は適切な次元を持つ任意の型の行列です (ここでは `mfu.nb_dof()`)、 mim は使用される積分方法を定義する `getfem::mesh_im` 型の変数です。 mfu は解の有限要素法を定義する必要がある `getfem::mesh_fem`

型の変数です。mfd は Lamé 係数が定義されている有限要素法を記述する `getfem::mesh_fem` 型の変数です (おそらくは mfu に等しい)。LAMBDA と MU はそれぞれ mfd の自由度に関する Lamé 係数の値のベクトルです。

ご用心: 線形弾性問題はベクトル問題であるため、mfu (`mf.set_qdim(Q)` を参照) の対象次元はメッシュの次元と同じでなければなりません。

ソース項、Neumann および Dirichlet 条件を構築するために、前のセクションと同じ関数を使用します。

12.3 混合有限要素法によるストークス問題

混合項の組み立て $B = -\int p \nabla \cdot v$ は:

```
getfem::asm_stokes_B(MATRIX &B, const mesh_im &mim,
                    const mesh_fem &mf_u, const mesh_fem &mf_p);
```

12.4 質量行列の構築

2つの有限要素間の質量行列の構築:

```
getfem::asm_mass_matrix(M, mim, mf1, mf2);
```

同じ関数の境界上の質量行列を得ることも可能です:

```
getfem::asm_mass_matrix(M, mim, mf1, mf2, nbound);
```

nbound は `mim.linked_mesh()` の領域インデックス、または `mesh_region` オブジェクトです。

第 13 章

任意の量の補間

一旦解が計算されると、例えば後処理のための補間関数を用いて解に対する任意の量を抽出することは簡単にできます。

13.1 基本補間

`getfem/getfem_interpolation.h` はあるメッシュ/有限要素法から他のメッシュ/他の Lagrange 有限要素法上の解を補間する関数 `getfem::interpolation(...)` を定義しています:

```
getfem::interpolation(mf1, mf2, U, V, extrapolation = 0);
```

ここで `mf1` は `getfem::mesh_fem` 型の変数でソースフィールド `U` が定義されている有限要素法を記述します。 `mf2` は `U` が補間される有限要素法です。 `extrapolation` はオプションのパラメータです。値は外挿を許さない `0`、境界近くの外点の外挿のための `1`、すべての外点の外挿のための `2` (コストがかかるかもしれない) です。

`U` の次元は `mf1.nb_dof()` の倍数でなければならず、補間されたデータ `V` は正しいサイズ (`mf2.nb_dof()` の倍数) でなければなりません。

... 重要:

```
``mf2`` should be of Lagrange type for the interpolation to make sense but the meshes linked to ``mf1`` and ``mf2`` may be different (and this is the interest of this function). There is no restriction for the dimension of the domain (you can interpolate a 2D mesh on a line etc.).
```

同じ有限要素法の間で複数の補間を実行する必要がある場合は、次の関数を使用する方が効率的です。

```
getfem::interpolation(mf1, mf2, M, extrapolation = 0);
```

ここで、 M は、補間を表す線形写像（すなわち、 $V = MU$ ）を満たす行行列です。行列は正しい次元（`mf2.nb_dof()` x `mf1.nb_dof()`）を持つ必要があります。この行列が構築されると、単純な行列乗算で補間が行われます:

```
gmm::mult(M, U, V);
```

13.2 高レベル弱形式言語に基づく補間

弱形式言語と補関数により、いくつかのフィールドで任意の式を抽出することが可能です。

これは特に `model` オブジェクト専用です（ただし、`ga_workspace` オブジェクトでも使用できます）。たとえば、`md` がいくつかの定義された変数 u （ベクトル）と p （スカラー）を含む有効なオブジェクトである場合、`p*Trace(Grad_u)` のような式を Lagrange 有限要素法で補間することができます。結果の式は、スカラ、ベクトル、またはテンソルになります。結果のベクトルのサイズは自動的に適合されます。

高水準の一般補関数はファイル `getfem/getfem_generic_assembly.h` で定義されています。

同じメッシュ上のラグランジェ有限要素法の補間、ポイントまたは `getfem::im_data` オブジェクト上の点群の補間に対応する補関数があります。

ラグランジェ有限要素法:

```
void getfem::ga_interpolation_Lagrange_fem(workspace, mf, result);
```

`workspace` は異なる変数とデータを格納することを目的とした `getfem::ga_workspace` オブジェクトです（参照 [任意の項を計算する - 高水準の汎用的な構築手順 - 弱形式言語](#)）、`mf` は `getfem::mesh_fem` オブジェクトは補間を行う Lagrange の有限要素法を表し、`result` は補間を格納する `beot::base_vector` です。ワークスペースには、補間される式が含まれている必要があります。

```
void getfem::ga_interpolation_Lagrange_fem(md, expr, mf, result, rg=mesh_region::all_convexes());
```

`md` は `getfem::model` オブジェクト（変数とデータを含んでいます）であり、`expr` (`std::string` オブジェクト) は補間される式で、`mf` は補間を行う Lagrange 有限要素法を表す `getfem::mesh_fem` オブジェクトです。`result` は補間が格納されるベクトルであり、`rg` は任意のメッシュ領域です。

点群の補間:

```
void getfem::ga_interpolation_mti(md, expr, mti, result, extrapolation = 0, rg=mesh_region::all_convexes());
```

`md` は `getfem::model` オブジェクト（変数とデータを含む）で、`expr` (`std::string` オブジェクト) は補間される式で、`mti` は点群を格納する `getfem::mesh_trans_inv` オブジェクト（`getfem/getfem_interpolation.h` 参照）、`result` は補間が格納されているベクトル、

extrapolation は外側の点の外側のメッシュでフィールドを外挿するオプションです。rg はオプションのメッシュ領域で、nbpnts はオプションの最大点数です。

im_data オブジェクトの補間（積分法の Gauss 点で）:

```
void getfem::ga_interpolation_im_data(md, expr, im_data &imd,
    base_vector &result, const mesh_region &rg=mesh_region::all_convexes());
```

ここで md は getfem::model オブジェクト（変数とデータを含んでいます）、expr (std::string オブジェクト) は補間される式、imd は積分法（getfem/getfem_im_data.h 参照）を参照する getfem::im_data オブジェクト、result は補間が格納されているベクトル、rg オプションのメッシュ領域です。

第 14 章

GetFEM++ に新しい有限要素法を組み込む

基本的には、参照要素に要素を記述するだけで十分です。すなわち各自由度の各基底関数を記述するということです。本質的にベクトル要素がサポートされています（例えば、Nedelec と Raviart-Thomas 要素）。実要素に依存する自由度の追加の線形変換と等価ではない有限要素法（ベクター要素、Hermite 要素などの... *GetFEM++* 専門の非 τ 方程式）を記述する必要があります（例として Argyris 要素の実装を参照）。

詳細は *dp* を読んでください。実際の実装は `getfem/getfem_fem.h`、`getfem_fem.cc` を参照してください。

第 15 章

GetFEM++ に新しい近似積分法を組み込む

perl スクリプトは記述ファイルから新しい立方体法を自動的に組み込みます。このような記述ファイル（拡張子は .IM）はディレクトリ cubature にあります。たとえば、IM_TETRAHEDRON(5) の場合、次のファイルはメソッドを記述しています:

```
NAME = IM_TETRAHEDRON(5)
N = 3
GEOTRANS = GT_PK(3,1)
NBPT = 4
0, 0.25, 0.25, 0.25, 0.008818342151675485
1, 0.31979362782962991, 0.31979362782962991, 0.31979362782962991, 0.011511367871045398
1, 0.091971078052723033, 0.091971078052723033, 0.091971078052723033, 0.01198951396316977
1, 0.056350832689629156, 0.056350832689629156, 0.44364916731037084, 0.008818342151675485
NBF = 4 IM_TRIANGLE(5)
IM_TRIANGLE(5)
IM_TRIANGLE(5)
IM_TRIANGLE(5)
```

ここで、NAME は、*GetFEM++*（定数整数パラメータが許される）のメソッド名です。N は次元、GEOTRANS は *GetFEM++* の有効な幾何変換を記述します。この幾何学変換は、積分法が記述されている基準要素を定義するだけです。NBPT は積分節点定義の数です。積分節点定義には積分節点の総数が NBPT より大きい対称定義が含まれています。

積分節点定義の構成:

- 整数: 0 = 対称性なし、1 = 完全対称 (3 角形の場合は x6、四角形の場合は x4、4 面体の場合は x24 ...)、
- 積分節点の N 座標、
- 荷重。

NBF は参照要素の面の数です (GEOTRANS に対応する必要があります)。次に、既存の積分方法を各面 (各面に

1 つずつ) に設定します。これは、境界を積分するために必要です。

ファイル形式は [EncyclopCubature] からインスピレーションを受けています。

第 16 章

レベル集合法、拡張有限要素法 (Xfem)、仮想領域有限要素法、切断有限要素法

v2.0 以降、*GetFEM++* は拡張有限要素法 (Xfem) と仮想領域有限要素法を切断有限要素法でサポートするために、いくつかの機能を提供しています。これらのツールのほとんどは、当初、Julien Pommier によって [LA-PO-RE-SA2005] に発表された研究をもとにして主に開発されました。

[SU-CH-MO-BE2001] で提案されているように、レベル集合法の使用に基づいて、実装はかなり大きな汎用性を持ち、多数のレベル集合法を同時に使用できます。

ジャンプの離散化のための拡張有限要素法 (Xfem) の実装は、[HA-HA2004] の手法に従いますが、実装中にこの作業についての知識はありませんでした。ゆえに、レベル集合法全体のジャンプを表す自由度がありません。代わりに、自由度は、レベル集合法の各側の変位を表します。これは、1つの要素の中に連続な2つ以上の異なるゾーンが存在する可能性があるため、互いに交差するレベル集合法の存在下ではどのような場合にも不可欠です。

仮想ドメイン法の切断有限要素法は、*GetFEM++* で初めて使用されています。かなり単純な安定化戦略が提案されている [HA-RE2009] に掲載された研究を使用しています。ここでも、話題の E. Burman and P. Hanbo [bu-ha2010] の存在を知る前に実装が行われています。

拡張有限要素法 (Xfem) は、J. Larys の PhD 研究によるもの (例えば、[LA-RE-SA2010] を参照) と E. Chahine のもの (例えば、[CH-LA-RE2011], [NI-RE-CH2011] を参照) と S. Amdouni のもの (例えば、[AM-MO-RE2014], [AM-MO-RE2014b] を参照) と M. Fabre のもの (例えば、[Fa-Po-Re2015] を参照) によってツールを充実させています。

重要: 下記のすべてのツールは、システムに `qhull` パッケージがインストールされている必要があります。このパッケージは広く入手可能です。このライブラリは、任意の次元の凸包と Delaunay3 角形分割を計算します。

プログラム `tests/crack.cc`、`interface/tests/matlab/crack.m` と `interface/tests/python/crack.py` はこれらのツールの使用例です。

16.1 レベル集合法の表現

いくつかの構造は、メッシュ上の区分的多項式関数によって定義されたレベル集合法関数を操作するために定義されています。ファイル `getfem/getfem_levelset.h` レベル集合法は、メッシュ上のある程度の Lagrange 有限要素法に定義された関数で表されます。新しい `getfem::level_set` を定義するコンストラクタは次の通りです。

```
getfem::level_set ls(mesh, degree = 1, with_secondary = false);
```

ここで、`mesh` は有効な `getfem::mesh` メッシュ型です、`degree` は多項式の次数 (1 はデフォルト値) です、また `with_secondary` はデフォルト値が `false` のブール値です。次レベル集合法は、降伏を表すために使用されます ($p(x)$ がプライマリレベル集合法関数であり、 $s(x)$ がセカンダリレベル集合法関数である場合、 $p(x) = 0$ と $s(x) \leq 0$: 2 次的役割はクラックを区切ることです。)

各レベル集合法関数は、`mesh_fem mf` とこれ以上の自由度値はベクトル内にあります。オブジェクト `getfem::level_set` は `mesh_fem` と対応する関数のための自由度のベクトルを含みます。 `ls.value(0)` メソッドは、プライマリレベル集合法関数の自由度のベクトルを返します。これにより、これらの値を設定することができます。 `ls.value(1)` メソッドは、セカンダリレベル集合法関数の自由度のベクトルを返します。メソッド `ls.get_mesh_fem()` は、`getfem::mesh_fem` オブジェクトです。

アプリケーションでは、(例えば、再初期化のため) Hamilton-Jacobi 方程式により、レベル集合法関数はしばしば進展することに注意してください。Hamilton-Jacobi 方程式の近似に使用できる [純対流法](#) を参照してください。

16.2 レベル集合法によるメッシュ切断

1 つまたは複数のレベル集合法にわたって不連続であるフィールドを表現するための適合化された積分方法および有限要素法を計算するために、適切な数の事前計算がメッシュレベルで行われなければなりません。 `getfem/getfem_mesh_level_set.h` では、これらの事前計算を処理するオブジェクト `getfem::mesh_level_set` が定義されています。このオブジェクトのコンストラクタは次のとおりです。

```
getfem::mesh_level_set mls(mesh);
```

ここで `mesh` は `getfem::mesh` 型の有効なメッシュです。メッシュがレベル集合法で切断されていることを示すためには、メソッド `mls.add_level_set(ls)` を呼び出さなければなりません。ここで、`ls` は `getfem::level_set` 型のオブジェクトです。任意の数のレベル集合法を追加することができます。オブジェクトを初期化するか、レベル集合法関数の値が変更されたときにオブジェクトを実現するには、`mls.adapt()` メソッドを呼び出さなければなりません。

特に、レベル集合法によって切断された各要素の細分化は、単純化によって行われます。切断メッシュ全体は一般にコンフォーマルではないことに注意してください。

切断メッシュは例えば後処理のために取得できます、切断メッシュで満たされている m により `mls.global_cut_mesh(m)` を呼び出します。

16.3 統合された積分方法

レベル集合法にわたって不連続なフィールドの場合、積分方法を適合させる必要があります。オブジェクト `getfem::mesh_im_level_set` は `getfem/getfem_mesh_im_level_set.h` ファイル内で定義されており、レベル集合法で切り取られた要素の合成積分法が定義されています。このオブジェクトのコンストラクタは次のとおりです。

```
getfem::mesh_im_level_set mim(mls, where, regular_im = 0, singular_im = 0);
```

ここで、`mls` は `getfem::mesh_level_set` 型のオブジェクトです、`where` は以下の可能な値がある列挙型です。

- `getfem::mesh_im_level_set::INTEGRATE_INSIDE` ($p(x) < 0$ 上で積分)、
- `getfem::mesh_im_level_set::INTEGRATE_OUTSIDE` ($p(x) > 0$ 上で積分)、
- `getfem::mesh_im_level_set::INTEGRATE_ALL`、
- `getfem::mesh_im_level_set::INTEGRATE_BOUNDARY` ($p(x) = 0$ と $s(x) \leq 0$ 上で積分)

引数 `regular_im` は、`pintegration_method` 型でなければなりません。また、レベル集合法で切断された要素の複合積分の各サブシンプレックスに適用される積分方法になります。オプションの `singular_im` は `pintegration_method` 型でなければならず、クラックの特異関数に使われます。クラックチップとの頂点を共有するサブシンプレックスに適用されます (特定の積分方法 `IM_QUASI_POLAR(...)` がこの目的に適しています)。

オブジェクト `getfem::mesh_im_level_set` は古典的な `getfem::mesh_im` として使うことができます。オブジェクト (例えば、`mim.set_integration_method(...)` メソッドは、レベル集合法によって切断されていない要素の積分法を設定することができます)。

オブジェクトを初期化するか、レベル集合法関数の値が変更されたときにそれを実現するには、`mim.adapt()` メソッドを呼び出さなければなりません。

`getfem::mesh_level_set` で複数のレベル集合法が宣言されている場合、オブジェクトを使用すると、メソッドを使用してより正確に積分ドメインを設定することができます:

```
mim.set_level_set_boolean_operations("desc");
```

“`desc`” は、積分ドメインを定義するブール演算の記述を含む文字列です。構文は単純です。たとえば、3つの異なるレベル集合法がある場合、

“ $a*b*c$ ” は、各レベル集合法で定義されたドメインの共通部分です（この関数が呼び出されない場合のデフォルト動作です）。

“ $a+b+c$ ” は、そのドメインの和集合です。

“ $c-(a+b)$ ” は、他の 2 つの和集合との第 3 レベル集合法の差のドメインです。

“! a ” は a のドメインの補集合（すなわち、 $a(x)>0$ の部分のドメイン）です

第 1 のレベル集合法は常に “ a ” と呼ばれ、第 2 のレベル集合法は “ b ” などと呼ばれます。

16.4 切断有限要素法

[bu-ha2010] に記載されているような切断有限要素法、すなわち、より小さな実領域に限定された架空の領域上の有限要素の実施は、これまでのツールおよび主として適応された積分法を使用することによって可能です。 *GetFEM++* にいくつかのテストプログラムの例があります。たとえば `interface/tests/python/demo_fictitious_domain.py` または `interface/tests/matlab/demo_fictitious_domain.m` を参照してください。

この文脈において、未知の有限要素フィールドを、対応する形状関数サポートが実ドメインとの交差を有する自由度に制限する必要があることが多い。これは、`partial_mesh_fem` オブジェクトを使って行うことができます。たとえば、`interface/tests/matlab/demo_structural_optimization.m` を参照してください。

Dirichlet 状態を適用する場合など、実際の領域との交差が非常に小さい要素による最終的なロック現象を処理するために、しばしば安定化技術を考慮する必要があることに注意してください。たとえば、[bu-ha2010]、[HA-RE2009]、[Fa-Po-Re2015] を参照してください。

16.5 いくつかのレベル集合法にわたる不連続なフィールド

`getfem::mesh_fem_level_set` オブジェクトはファイル `getfem/getfem_mesh_fem_level_set.h` で定義されています。それは `getfem::mesh_fem` オブジェクトから派生しており、同じ方法で使用できます。これは、レベル集合法全体に（レベル集合法の任意の数を扱うことができる）不連続の有限要素法を定義します。コンストラクタは次のとおりです。

```
getfem::mesh_fem_level_set mfls(mls, mf);
```

`mls` は `getfem::mesh_level_set` 型の有効なメッシュで、`mf` は `getfem::mesh_fem` 型のオブジェクトです。これは、レベル集合法によって切断されない要素に使用される有限要素法を定義します。

オブジェクトを初期化するか、レベル集合法関数の値が変更されたときにオブジェクトを実現するには、`mfls.adapt()` メソッドを呼び出さなければなりません。

非連続的なフィールドを表現するために、有限要素法は、いくつかの **Heaviside** 関数と、`mf` で表される有限要素法の形状関数の積である不連続関数に強化されています（詳細については [\[HA-HA2004\]](#) および [\[Xfem\]](#) を参照）。

16.6 拡張有限要素法 (Xfem)

拡張有限要素法 (Xfem) ([\[Xfem\]](#) を参照) は、いくつかの **Heaviside** 関数 (オブジェクト “`getfem :: mesh_fem_level_set`” によって行われます) だけでなく、クラック先端での漸近的な変位による豊かさでも構成されています。[\[Xfem\]](#) のような亀裂先端を含む要素のみの高度化、[\[LA-PO-RE-SA2005\]](#) または [\[Be-Mi-Mo-Bu2005\]](#) のような固定されたサイズの領域の高度化は、[\[CH-LA-RE2008\]](#) または [\[NI-RE-CH2011\]](#) のようなカットオフ関数を用いた高度化または高度化領域と非高度化領域との間の完全一致条件を漸近変位で高度化させるいくつかの方法が [\[CH-LA-RE2011\]](#) などのようにあります。`Getfem` は、すべての可能性を簡単に実装するための最大限の柔軟性に欠けていました。主に有限要素法自体の変換であるため、いくつかの豊富な有限要素を生成するための 2 つのツールが定義されています。

```
getfem::mesh_fem_product mf_asympt(mf_part_unity, mf_sing)
getfem::mesh_fem_sum mf_sum(mf1, mf2)
```

`getfem::mesh_fem_global_function` オブジェクト (`src/getfem/getfem_mesh_fem_global_function.h` を参照) のために、`mf_sing` はグローバルな ‘有限要素法’ でなければなりません。実際にグローバル関数の集合 (カットオフ関数の有無にかかわらず) と `mf_part_unity` は基本的なスカラー有限要素法です。結果として得られる `getfem::mesh_fem_product` は、与えられた 2 つの有限要素法の形状関数のすべての積の線形結合であり、メソッド `mf_asympt.set_enrichment(enriched_dofs)` により与えられる有限要素法の自由度の部分集合に制限されます。

漸近的濃縮が定義されると、オブジェクト `getfem::mesh_fem_sum` は 2 つの有限要素法の直和を生成することを可能にします。たとえば、**Heaviside** 関数 (`getfem::mesh_fem_level_set` オブジェクト) は漸近的な高度化によって充実したものの例です。

これらのツールの使用例については、`interface/tests/matlab/demo_crack.m`、`interface/tests/python/demo_crack.py` または `tests/crack.cc` を参照してください。

さらに、弱形式言語は、レベル集合法全体の任意のフィールドまたは任意のフィールドのジャンプを考慮に入れることを可能にする 2 つのコマンド `Xfem_plus` および `Xfem_minus` を定義します (`ud-gasm-high_xfem` を参照)。この先験的な部分は、任意のインタフェース則を容易に書くことを可能にしていることです。

また、`src/getfem/getfem_crack_sif.h` の中には、2D の応力拡大係数を計算するためのいくつかの手順があります (均一な等方線形弾性に制限されています)。

16.7 ポスト処理

レベル集合法の側でのみ解を表現するツールや、(Xfem 近似のための) 不連続性も考慮した両方のツールがあります。

切断メッシュ `mls` (すなわち、`getfem::mesh_level_set` オブジェクト) が使われているとき、コマンドで全てのサブ要素の集合を得ることができます。

```
mls.global_cut_mesh(mcut);
```

`mcut` は、サブ要素で塗りつぶされる空のメッシュでなければなりません。生成されたメッシュは、すべての要素のサブメッシュが要素のエッジ/面でコンフォーマルでないという意味で、非正規のメッシュであることに注意してください。しかしながら、このメッシュ上の Lagrange 有限要素法を相互に連結し、不連続なフィールドを正しく表現するために後処理を行うことが可能です。

仮想領域有限要素法が使用されているときに、解の着目部分だけを表す別の手段は、等値レベル集合法 ([メッシュスライスの生成](#) 参照) で定義されたメッシュスライスを使用することです。

`interface/tests/matlab/demo_crack.m`、`interface/tests/python/demo_fictitious_domain.py` と `interface/tests/matlab/demo_structural_optimization.m` を参照してください。

第 17 章

非適合メッシュ上の有限要素法の補間

特殊な有限要素法は、`getfem/getfem_interpolated_fem.h` で定義されています。これは実際の有限要素法ではありませんが、別のメッシュで定義された有限要素法を補間する `pseudo-fem` です。異なるメッシュで定義された有限要素法を使用して行列をアセンブルする必要がある場合は、”interpolated fem” を使用することができます。

```
getfem::new_interpolated_fem(getfem::mesh_fem mf, getfem::mesh_im mim);
```

有限要素法の各基底関数は補間されなければならないので、そのような計算は重い手順である可能性があります。デフォルトでは、補間された有限要素法オブジェクトは補間データを格納します。

補間は積分方法 `mim` の各 **Gauss** 点で行われるので、構築手順でこれらの積分方法を使わなければなりません。

たとえば、2つの異なるメッシュで定義された2つの異なる有限要素法の間で質量行列を計算する必要がある場合、これは2番目の有限要素法を補間するコードの例です。`mf` が有限要素法を記述し、`mim` が選択された積分法であると仮定して、最初の有限要素法メッシュ上で

```
getfem::mesh_fem mf_interpole(mfu.linked_mesh());
pfem ifem = getfem::new_interpolated_fem(mf, mim);
dal::bit_vector nn = mfu.convex_index();
mf_interpole.set_finite_element(nn, ifem);
getfem::asm_mass_matrix(SM1, mim, mfu, mf_interpole);
del_interpolated_fem(ifem);
```

`ifem` が指すオブジェクトには、補間に関するすべての情報が含まれています。それは多くのメモリを使用する可能性があります。`pfem` はスマートポインタ (`boost intrusive_ptr`) であるため、補間された有限要素法は、最後のポインタが破棄されると自動的に破棄されます。より正確な精度を得るには、その次数を増やすのではなく、積分メソッドを（たとえば、`IM_STRUCTURED_COMPOSITE` を使って）改良する方が良いでしょう。

17.1 異なるメッシュの混合メソッド

説明する...

17.2 モルタル法

説明する...

第 18 章

L^2 と H^1 ノルムの計算

getfem/getfem_assembling.h は、 L^2 と H^1 ノルムを計算するため関数を定義します。以下の関数は、異なるノルムを計算します

```
getfem::asm_L2_norm(mim, mf, U, region = mesh_region::all_convexes());
getfem::asm_H1_semi_norm(mim, mf, U, region = mesh_region::all_convexes());
getfem::asm_H1_norm(mim, mf, U, region = mesh_region::all_convexes());
```

ここで mim は getfem::mesh_im 積分のために使われ、mf は getfem::mesh_fem で、解が定義される有限要素法を記述します、U は mf の自由度の解のベクトルであり、region はノルムが計算されるメッシュ領域を指定する任意のパラメータです。U のサイズは mf.nb_dof() でなければなりません。

2 つの解を比較するには、1 つの *mesh_fem* を補間するよりも、次の関数を使用する方が簡単で速くなるのがよくあります。

```
getfem::asm_L2_dist(mim, mf1, U1, mf2, U2, region = mesh_region::all_convexes());
getfem::asm_H1_dist(mim, mf1, U1, mf2, U2, region = mesh_region::all_convexes());
```

これらの関数は L^2 と H^1 ノルムの $u_1 - u_2$ を返します。

第 19 章

導関数の計算

ファイル `getfem/getfem_derivatives.h` は解の勾配を計算する次の関数を定義します:

```
getfem::compute_gradient (mf1, mf2, U, V);
```

ここで `mf1` は `mesh_fem` 型の変数で、解が定義されている有限要素法を記述します、`mf2` は勾配を計算する有限要素法を記述し、`U` は解を表すベクトルであり、サイズは `mf1.nb_dof()` である必要があります、`V` は勾配が計算されるベクトルでありサイズが $N * mf2.nb_dof()$ で、領域の次元が N でなければなりません。

第 20 章

解の出力と表示

`getfem` の計算結果を見るには、基本的に 4 つの方法があります。

- `matlab` インターフェースを使用した `Matlab` による出力
- オープンソースの `Mayavi` やその他の `VTK` ファイルビューア
- オープンソースプログラムの `OpenDX`
- オープンソースのプログラム `Gmsh`

出力できるオブジェクトは、`mesh`、`mesh_fem`、`stored_mesh_slice` オブジェクトです。

20.1 Matlab インターフェース用の `mesh` と `mesh_fem` オブジェクトの保存

`Matlab` インターフェースをインストールしている場合は、単に `mesh_fem::write_to_file` を実行し、解をプレーンテキストファイルとして保存し、`Matlab` にロードしてください。例えば、`mesh_fem mf` 上に解 `U` を持っている場合、

```
std::fstream f("solution.U", std::ios::out);
for (unsigned i=0; i < gmm::vect_size(U); ++i)
    f << U[i] << "\verb+\n";

// when the 2nd arg is true, the mesh is saved with the |mf|
mf.write_to_file("solution.mf", true);
```

と入力し、`matlab` の下で以下を実行します。

```
>> U=load('solution.U');
>> mf=gfMeshFem('load','solution.mf');
>> gf_plot(mf,U,'mesh','on');
```

詳細については、`getfem-matlab` インタフェースのドキュメントを参照してください。

他の2つのファイルフォーマット、すなわち `VTK` ファイルフォーマット、`OpenDX` ファイル形式と `Gmsh` 後処理ファイル形式の両方で、`getfem::mesh` または `getfem::mesh_fem` のどちらかを出力するだけでなく、より汎用性の高い `getfem::stored_mesh_slice` を出力します。

使用例は `tests` ディレクトリの例にあります。

20.2 メッシュスライスの生成

`GetFEM++` は “slicer” オブジェクトを提供します。このオブジェクトはポストメッシュとソリューションからの処理データです。これらのスライサーは、ファイル `getfem/getfem_mesh_slicers.h` で定義され `mesh` を利用します (時には解フィールド) を入力して、平面との交差、メッシュの抽出境界、各凸の改善、等値面の抽出、等これらのスライサーの出力は、`getfem::stored_mesh_slice` オブジェクトに格納することができます (`getfem/getfem_mesh_slice.h` を参照)。 `stored_mesh_slice` オブジェクトは高速補間を伴う非等角メッシュ上の `P1` 不連続 FEM として、スライスがセグメント、3 角形、4 面体で作られているので、元のメッシュの凸包は常に単純化されています。

すべてのスライサー操作は `getfem::slicer_action` から継承します。新しいスライス操作を作成するのは非常に簡単です。スライサーの例は (いくつかは `getfem::mesh_slice_cv_dof_data_base` を使用しています。これは `mesh_fem` `mf` と `mesh_fem` 上のフィールド `U` への単なる参照です)。

```
getfem::slicer_none()
```

空のスライサー

```
getfem::slicer_boundary(const mesh &m, ldots)
```

メッシュの境界を抽出します

```
getfem::slicer_apply_deformation(mesh_slice_cv_dof_data_base &)
```

変形をメッシュに適用する、`mesh_fem` 上の変形フィールド

```
getfem::slicer_half_space(base_node x0, base_node n, int orient)
```

半分の体積 (`orient = -1` または `+1` の場合) または面 (`orient = 0` の場合) でメッシュをカットする、`x0` は面節点で、`n` は面の法線。

```
getfem::slicer_sphere(base_node x0, scalar_type R, int orient)
```

内部 (`orient = -1`)、境界 (`orient = 0`) または外部 (`orient = +1`) または中心 `x0` と半径 `R` でカットします。

`getfem::slicer_cylinder` (base_node $x0$, base_node $x1$, scalar_type R , int $orient$)

軸 ($x0$, $x1$) の円柱の内側/境界/外側のスライスで半径 R としている。

`getfem::slicer_isovalues` (const mesh_slice_cv_dof_data_base& mfU , scalar_type val , int $orient$)

スカラーフィールド mfU と val で定義された等値面でカットします。単純なものをまとめておきます
 $u(x) < val$ ($orient = -1$), $u(x) = val$ ($orient = 0$ または $u(x) > val$)。

`getfem::slicer_mesh_with_mesh` (const mesh& $m2$)

メッシュ $m2$ の凸包で凸包をカットします。

`getfem::slicer_union` (const slicer_action & sA , const slicer_action & sB)

2つのスライサー操作の出力をマージします。

`getfem::slicer_intersect` (slicer_action & sA , slicer_action & sB)

2つのスライサー操作の出力を交差させます。

`getfem::slicer_complementary` (slicer_action & s)

スライサー操作の補完を返します。

`getfem::slicer_build_edges_mesh` (mesh& $edges_m$)

そのスライスメッシュのエッジでメッシュ $edges_m$ を構築します。

`getfem::slicer_build_mesh` (mesh & m)

ある種の (まれな) 場合には、スライスからメッシュを構築すると便利かもしれません。しかし、(しばしばそうですが) メッシュが等角になります。

`getfem::slicer_build_stored_mesh_slice` (stored_mesh_slice& sl)

スライシング操作の出力を `stored_mesh_slice` オブジェクトに記録します。
`stored_mesh_slice::build(...)` を使うほうが同じ結果を得るには便利なことに留意してください。

`getfem::slicer_explode` (c)

重心に対してそれぞれの凸包を縮小または拡大します。

これらのスライサーを適用するには、`getfem::mesh_slicer`(mesh&) オブジェクトを作成し、`getfem::slicer_action` を `mesh_slicer::push_back_action(slicer_actio&)` と `mesh_slicer::push_front_action(slicer_action&)` 。スライシング操作は最終的に `mesh_slicer::exec(int nrefine)` により実行されます (または `mesh_slicer::exec(int nrefine, const mesh_region&cvlst)` を実行すると、メッシュのサブセット、またはその境界などに対する操作などを行うことができます)。

`nrefine` パラメータは非常に重要です。なぜなら、最終結果の”精度”はこの変数に依存するからです。最後のスライスで表現されるデータが線形幾何変換を持つ凸面の P1 データだけである場合、`nrefine=1` が正しい選択ですが、P2、P3、非線形変換などでは、スライス操作中に元のメッシュの各凸面を改良する方が良いです。ほ

とんどの視覚化プログラム (gmsh、mayavi、opendx、matlab など) は、非常に単純な構造 (線形セグメント/3 角形/4 面体に P1 不連続データを持つ) 上に有限要素フィールドを正確に表現することができます。

使用例 (メッシュ `m` の境界を半分にカットし、結果を `stored_mesh_slice` に保存します) :

```
getfem::slicer_boundary a0(m);
getfem::slicer_half_space a1(base_node(0,0), base_node(1, 0), -1);
getfem::stored_mesh_slice sl;
getfem::slicer_build_stored_mesh_slice a2(sl);
getfem::mesh_slicer slicer(m);
slicer.push_back_action(a1);
slicer.push_back_action(a2);
int nrefine = 3;
slicer.exec(nrefine);
```

スライシング操作中に `getfem::stored_mesh_slice` オブジェクトを構築するための、`stored_mesh_slice::build()` メソッドは明らかに `slicer_build_stored_mesh_slice` スライサーを使うよりも便利です:

```
getfem::stored_mesh_slice sl;
sl.build(m, getfem::slicer_boundary(m),
         getfem::slicer_half_space(base_node(0,0), base_node(1, 0), -1),
         nrefine);
```

これらのスライスを使用する最も簡単な方法は、それらを `vtk`、`opendx`、または `gmsh` に出力することです。 `getfem/getfem_export.h` には 3 つのクラスがあります: `getfem::vtk_export`、`getfem::dx_export`、および `getfem::pos_export` です。

20.3 `mesh` か `mesh_fem` またはスライスを VTK に出力します

まず、VTK データファイルの制限を知っておくことが重要です。各ファイルには、1 つのメッシュだけが含まれ、このメッシュにはスカラーフィールドが 1 つ、ベクトルフィールドが 1 つとテンソルフィールドが 1 つ (この順序で) 含まれます。VTK ファイルは、セグメント、3 角形、四角形、4 面体、6 面体のデータを扱うことができます。2 次の 3 角形、セグメントなどはサポートされていると言われてはいますが、スライスを構築するときは `nrefine=2` を使うのと同じです。VTK データファイルは、複数のタイプの要素 (例えば、3 角形や四角形などのメッシュ) を使用してメッシュをサポートしています。

例えば、`stored_mesh_slice` の `sl` が既にビルドされていると仮定します:

```
// an optional the 2nd argument can be set to true to produce
// a text file instead of a binary file
vtk_export exp("output.vtk");
exp.exporting(sl); // will save the geometrical structure of the slice
```

```
exp.write_point_data(mfp, P, "pressure"); // write a scalar field
exp.write_point_data(mfu, U, "displacement"); // write a vector field
```

この例では、スライス P と U のフィールドは、スライス 節点に書き込まれ、VTK フィールドに書き込まれます。ベクトルフィールドは常にスカラーフィールドの後に書かれます（テンソルフィールドは最終です）。

スライスを構築せずに mf を出力することもできます：

```
// an optional the 2nd argument can be set to true to produce
// a text file instead of a binary file
vtk_export exp("output.vtk");
exp.exporting(mfu);
exp.write_point_data(mfp, P, "pressure"); // write a scalar field
exp.write_point_data(mfu, U, "displacement"); // write a vector field
```

しかし、このアプローチでは vtk_export はそれぞれの VTK 要素型に mfu の凸面/有限要素法を指定します。そのため、VTK が 2 次より次数の大きい要素を扱わない場合、より高い次元の有限要素法の精度が失われます。

20.4 mesh か mesh_fem またはスライスを OpenDX に出力する

OpenDX のデータファイルは VTK よりも汎用性があり、より多くのメッシュ、これらのメッシュの任意の数のフィールドなどを格納することができます。しかし、それが可能なのは 1 次と 0 次の要素（セグメント、3 角形、4 面体、四角形など）です。そして、それぞれのメッシュは 1 つのタイプしか作成できません。要素の 3 角形と四角形を同じオブジェクトに混在させることはできません。その理由から、（複雑な要素が単純化されている）getfem::stored_mesh_slice オブジェクトを getfem::mesh_fem および getfem::mesh オブジェクトよりも使用します。

基本的な使い方は getfem::vtk_export と非常に似ています。

```
getfem::dx_export exp("output.dx");
exp.exporting(sl);
exp.write_point_data(mfu, U, "displacement");
```

さらに、getfem::dx_export は ‘.dx’ ファイルを再オープンして新しいデータを追加することができます。そこで、多くの時間ステップを保存すれば、OpenDX の中間結果を確認することができます。コンストラクタの雛形は次の通りです。

```
dx_export(const std::string& filename, bool ascii = false, bool append = false);
dx_export(std::ostream &os_, bool ascii = false);
```

複数の時間ステップを持つ使用例（tests/dynamic_friction.cc を参照）は次の通りです。

```
getfem::stored_mesh_slice sl;
getfem::dx_export exp("output.dx", false);
if (N <= 2) sl.build(mesh, getfem::slicer_none(), 4);
else      sl.build(mesh, getfem::slicer_boundary(mesh), 4);
exp.exporting(sl, true);

// for each mesh object, a corresponding ``mesh`` object will be
// created in the data file for the edges of the original mesh
exp.exporting_mesh_edges();

while (t <= T) {
  ...
  exp.write_point_data(mf_u, U0);
  exp.serie_add_object("deformation");
  exp.write_point_data(mf_vm, VM);
  exp.serie_add_object("von_mises_stress");
}
```

この例では、時間ステップごとに OpenDX の“時系列”が作成され、2つのデータフィールドが保存されます：“deformation”と呼ばれるベクトルフィールドと、“von_mises_stress”というスカラーフィールドです。

`dx_export::exporting_mesh_edges()` 関数はこれは、出力されたメッシュごとに、オリジナルのメッシュも（別の OpenDX メッシュに）出力されます。この例では、OpenDX は4つのデータフィールドにアクセスします：“deformation”、“deformation_edges”、“von_mises_stress”と“von_mises_stress_edges”。

`tests/dynamic_friction.net` はこれらのデータのための OpenDX プログラムの例です (`cd tests; dx -edit dynamic_friction.net`、メニュー“実行/シーケンサー”で実行してください)。

第 21 章

純対流法

純粋な対流を計算する方法は、ファイル `getfem/getfem_convect.h` で定義されています。関数の呼び出しは:

```
getfem::convect(mf, U, mf_v, V, dt, nt, option = CONVECT_EXTRAPOLATION);
```

ここで `mf` は `getfem::mesh_fem` 型の変数です、`U` は対流するフィールドを表すベクトル、`mf_v` は `getfem::mesh_fem` 速度場の場合、`V` は速度場の自由度ベクトルであり、`dt` は対流の擬似時間であり、`nt` は特性計算のための反復回数です。 `option` は境界条件がリエントラント対流である場合のオプションです。可能なものは `getfem::CONVECT_EXTRAPOLATION` (最も近い要素のフィールドの外挿) または `getfem::CONVECT_UNCHANGED` (境界の値の変更なし) です。

この方法は、偏微分方程式

$$\frac{\partial U}{\partial t} + V \cdot \nabla U = 0,$$

の時間間隔 $[0, dt]$ を積分します。

使用される方法は Galerkin-Characteristic のものです。この方法は無条件に安定しているものの、むしろ散逸的で非常に簡単なバージョンです。[ZT1989] および `convect` コマンドに関する Freefem ++ のドキュメントを参照してください。

定義されたメソッドは `mf` がその時点の純粋な Lagrange 有限要素法である場合にのみ機能します。原理は常微分方程式を解くことによって有限要素節点を逆方向に対流させることです。

$$\frac{dX}{dt} = -V(X),$$

各節点に対応する初期条件を有します。この対流は `nt` ステップで行われます。次に、対流節点上で解を補間します。

外挿をそれほど高コストにしないために、積 $dt \times V$ は大きすぎたはけません。

この方法は分割スキームと結合する対流優位問題を解決するために使用することができることに留意してください。

第 22 章

モデル記述と基本モデルブリック

GetFEM++ のモデル記述は複雑な線形または非線形 PDE 結合モデル上にいくつかの *fem* アプリケーションを素早く構築することができます。原則は、複雑な状況を記述するために組み立てることができる所定の要素を提案することです。ブリックは、方程式 (Poisson 方程式、線形弾性...) または境界条件 (Dirichlet、Neumann ...) または 2 つの変数間の任意の関係を記述することができます。ひとたび要素が書かれると、それは非常に異なる状況で使用することが可能です。これにより、生成されたコードの再利用と、成長するライブラリのブリックの実装が可能になります。可能な限り新しい要素の定義を容易にするための努力がなされている。要素は、主に、接線系での寄与によって定義されます。

このモデル記述は、*GetFEM++* の旧バージョンのモデル要素を進化させたものです。古いシステムと比較して、より柔軟で、より汎用的であり、モデル (マルチフィジックス) のカップリングをより簡単な方法で可能にし、新しい成分の書き込みを容易にします。進展する PDE のための時間積分スキームの記述も容易です。

モデル記述のカーネルはファイル `get.fem/get.fem_models.h` に含まれています。2 つの主要なオブジェクトは *model* と *brick* です。

22.1 model オブジェクト

`get.fem/get.fem_models.h` で定義されている *model* オブジェクトの目的は、PDE モデルをグローバルに記述することです。主に 2 つのリストがあります: 変数のリスト (*mesh_fem* オブジェクトに関連するかどうか) とデータ (これも *mesh_fem* オブジェクトに関連するかどうか) と項のリスト。*model* の役割はモジュールを調整しそれらに式の線形システムを生成させることです。モデルが線形の場合、これは単純に対応する *dof* の式の線形システムになります。モデルが非線形の場合、これは接線系になります。*model* オブジェクトは実数と複素数の 2 つのバージョンがあります。

model オブジェクトの宣言は:

```
getfem::model md(complex_version = false);
```

であり、コンストラクタのパラメータは、モデルが複素数を扱うか実数を扱うかを決定するブール値です。デフォルトは false で実数を扱うモデルになります。

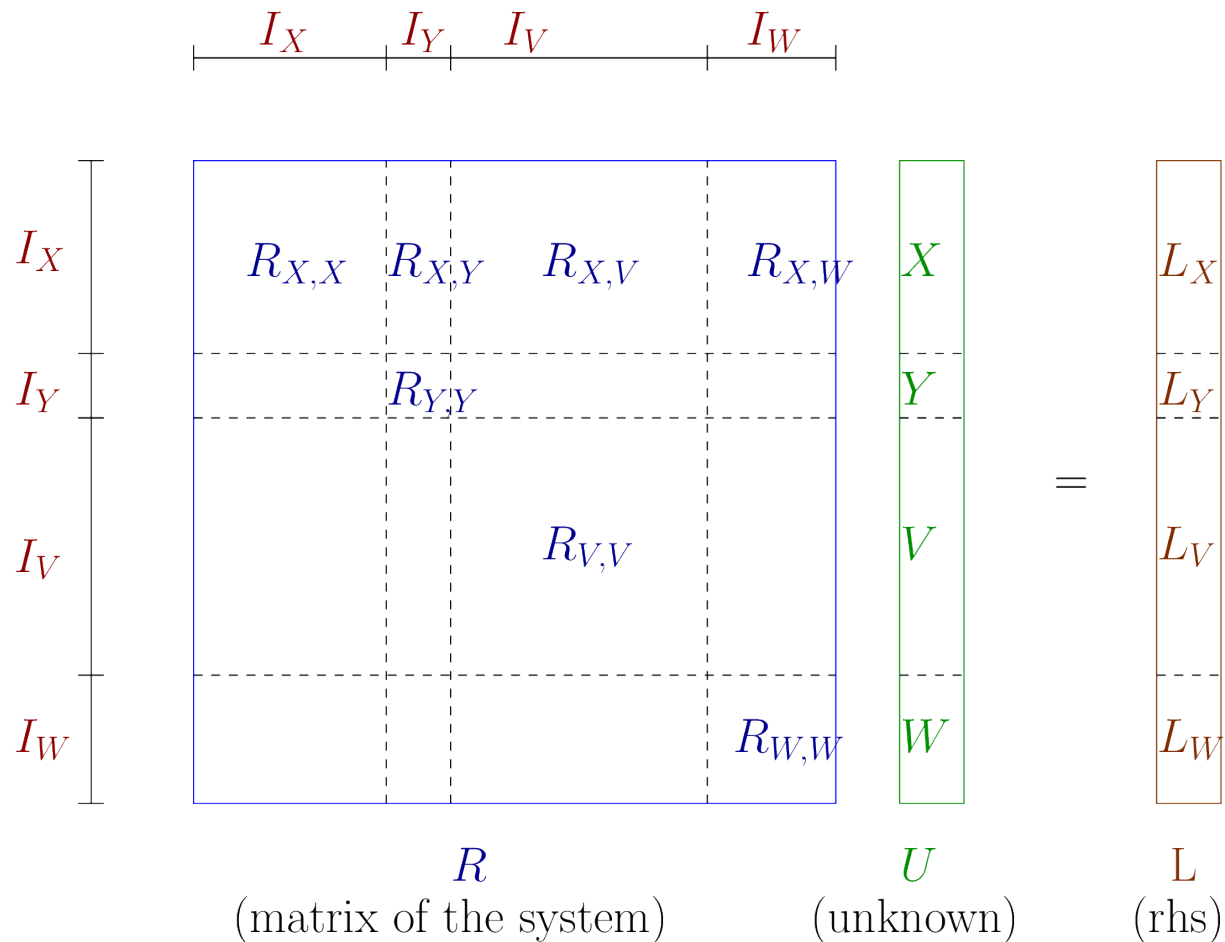


図 22.1 線形 (接線) システム

モデルにはさまざまな種類の変数/データがあります。変数はモデルの未知数です。これらは、モデルによって構築された (接線) 線形システムを解くことによって (汎用的に) 計算されます。一般に、モデルにはいくつかの変数があります。各変数は一定の大きさ (自由度の数) を持ち、さまざまな変数は英数字順にソートされ、グローバルな未知数 (図 線形 (接線) システム の U) を形成します。各変数はグローバルシステム内のこの変数に対応する自由度指標を表す $I = [n_1, n_2]$ という区間に関連付けられます。また、モデルはいくつかのデータを (変数と同じフォーマットで) 保存します。データと変数の違いは、データがモデルの未知数ではないということです。データの値は提供する必要があります。いくつかの (非線形モデル) の場合によっては、いくつかの変数はある種のデータとして考えることができます。変数とデータは 2 種類あります。それらは固定サイズを有することができ、または有限積分法 (有限積分法の自由度) に依存することができます。

例えば、図 線形（接線）システム に記述されている状況ではモデルには 4 つの変数、すなわち、 X, Y, V と W があります。model オブジェクトの役割は、線形システムを組み立てること、すなわち、各変数に対応する部分行列 ($R_{X,X}, R_{Y,Y}, R_{V,V}$, と $R_{W,W}$) を満たすことと 2 つの変数の間の項 ($R_{X,Y}, R_{X,V}, R_{W,V}, \dots$) を結合することです。この異なる寄与はモデルに追加された異なる項によって与えられます。

model オブジェクトの主な有用なメソッドは

m.is_complex()

モデルが実数または複素数の未知数とデータを扱うかどうかを示すブール値です。

add_fixed_size_variable(name, size, niter=1)

固定サイズの変数を追加します。name は変数を指定する文字列です。niter は変数のコピー数です。

add_fixed_size_variable(name, sizes, niter=1)

固定サイズの変数を追加します。name は変数を指定する文字列です。sizes は行列やテンソルの固定サイズ変数の次元のベクトルです。niter は変数のコピー数です。

add_fixed_size_data(name, size, niter=1)

固定サイズのデータを追加します。name はデータを指定する文字列です。niter はデータのコピー数です。

add_fixed_size_data(name, sizes, niter=1)

固定サイズのデータを追加します。name はデータを指定する文字列です。sizes は行列やテンソルの固定サイズ変数の次元のベクトルです。niter はデータのコピー数です。

add_initialized_fixed_size_data(name, V)

与えられたベクトル V で初期化された固定サイズのデータを追加します。name はデータを指定する文字列です。

add_initialized_scalar_data(name, e)

与えられたスカラー値 e で初期化されたサイズ 1 のデータを追加します。name はデータを指定する文字列です。

add_fem_variable(name, mf, niter=1)

有限積分法 mf の dofs である変数を追加します。name は変数を指定する文字列です。niter は変数のコピー数です。

add_fem_data(name, mf, niter=1)

有限積分法 mf の自由度数のデータ追加します。name はデータを指定する文字列です。niter はデータのコピー数です。

add_initialized_fem_data(name, mf, V, niter=1)

与えられたベクトル V で初期化された有限積分法 mf の自由度数のデータを追加します。name はデータを指定する文字列です。niter はデータのコピー数です。

add_multiplier (name, mf, primal_name, niter=1)

有限積分法 *mf* にリンクされ、第一変数 *primal_name* 上の特定の制約（例えば、Dirichlet 条件）のための乗数である特別な変数を追加します。最も重要なことは、線形独立の制約の集合だけを保持するために、*partial_mesh_fem* オブジェクトにより自由度がフィルタリングされることです。これを確実にするために、乗数と原変数を結ぶ項を持つ項への呼び出しが行われ、独立した制約を抽出するための特別なアルゴリズムが呼び出されます。このアルゴリズムは、境界乗数に最適化されています（`gmm::range_basis` を参照）。ポリウムマルチプライヤーに注意して使用してください。niter は変数のコピー数です。複素数項については、実部のみが乗数をフィルタリングするとみなされることに留意してください。

real_variable (name, niter=1)

変数またはデータのベクトル値にアクセスできます。実数バージョンです。

complex_variable (name, niter=1)

変数またはデータのベクトル値にアクセスできます。複素数バージョンです。

mesh_fem_of_variable (name)

定義されている変数の *mesh_fem* のリファレンスが与えられます。有限積分法変数でない場合は例外を返します。

real_tangent_matrix ()

接線行列へのアクセスが与えられます。実数バージョンです。最初に接線系の計算を行う必要があります。

complex_tangent_matrix ()

接線行列へのアクセスが与えられます。複素数バージョンです。最初に接線系の計算を行う必要があります。

real_rhs ()

線形システムの右辺ベクトルへのアクセスが与えられます。実数バージョンです。最初に接線系の計算を行う必要があります。

complex_rhs ()

線形システムの右辺ベクトルへのアクセスが与えられます。複素数バージョンです。最初に接線系の計算を行う必要があります。

22.2 brick オブジェクト

モデルブリックとはモデルの部分を表すオブジェクトです。このオブジェクトは PDE モデルの弱定式化でいくつかの積分項を表現するためにあります。model オブジェクトにはブリックのリストが含まれます。ブリックによって記述されたすべての項は最終的に線形システム（非線形問題のための接線系）を構築するために組み立てられます。例えばもし項 Δu が PDE モデル上に存在します（ u のラプラシアン）、次いで弱定式化は $\int_{\Omega} abla u \cdot abla v \, dx$ 項が含まれます。ここで v は u に対応する試行関数です。対応するブリックの役割は $\int_{\Omega} abla \varphi_i \cdot abla \varphi_j \, dx$ 項を

アセンブルすることです、 φ_i と φ_j は u を記述する有限積分法の形状関数です。この項は `model` オブジェクトによって変数 u に対応する対角ブロック上の全体線形システムに追加されます。したがって、ブリックの唯一の役割は `model` オブジェクトがそれを求めるときに対応する構築プロシーダを呼び出すことです。このように線形項のための項の構築は非常に簡単です。

基本的には、`brick` オブジェクトは `getfem/getfem_models.h` で定義された `virtual_brick` オブジェクトから派生し、実数項か固有の複素数項かによって `asm_real_tangent_terms` または `asm_complex_tangent_terms` メソッドを再定義する必要があります。

22.3 新しい項を作る方法

まず最初に注意しておきたいのは、新しい項の設計は、既存の項ではカバーされず、汎用的な構築項の幅広いアクセシブルな項（複雑な結合条件を含む）でカバーされていない特別な項に対してのみ必要である点に注意してください（汎用的な構築ブリック）。

設計精神によれば、項は追加のデータをできるだけ保存しないようにすべきです。ブリックのパラメータはモデルの変数とデータに含める必要があります。例えば、線形弾性ブリックのパラメータは弾性係数です。この係数はモデルの何らかのデータでなければなりません。項が `model` オブジェクトによって呼び出されると、変数とデータのリストが項に与えられます。あらかじめ定義された項の大部分はデータを格納しません。これにより、そのような項を一度だけインスタンス化することができます。

ラプラシアン項に対応する項の例は次のとおりです（他の例は標準的な項を含むファイル `getfem_models.cc` で確認できます）。

```
struct my_Laplacian_brick: public getfem::virtual_brick {

    void asm_real_tangent_terms(const getfem::model &md, size_type ib,
                               const getfem::model::varnamelist &varl,
                               const getfem::model::varnamelist &datal,
                               const getfem::model::mimlist &mims,
                               getfem::model::real_matlist &matl,
                               getfem::model::real_veclist &vecl,
                               getfem::model::real_veclist &vecl_sym,
                               size_type region, build_version nl) const {
        GMM_ASSERT1(matl.size() == 1,
                    "My Laplacian brick has one and only one term");
        GMM_ASSERT1(mims.size() == 1,
                    "My Laplacian brick need one and only one mesh_im");
        GMM_ASSERT1(varl.size() == 1 && datal.size() == 0,
                    "Wrong number of variables for my Laplacian brick");

        const getfem::mesh_fem &mf_u = md.mesh_fem_of_variable(varl[0]);
        const getfem::mesh_im &mim = *mims[0];
```

```

gmm::clear(matl[0]);
getfem::asm_stiffness_matrix_for_homogeneous_laplacian
(matl[0], mim, mf_u, region);
}

my_Laplacian_brick(void)
{ set_flags("My Laplacian brick", true /* linear */,
           true /* symmetric */,
           true /* coercivity */,
           true /* real version defined */,
           false /* no complex version*/);
}
};

```

項のコンストラクタは `set_flags` メソッドを呼び出す必要があります。このメソッドの最初のパラメータは項の名前です（これにより、モデルの項をリストしその識別を容易にすることができます）。その他のパラメータは、それぞれ次のフラグです。

- 項項がすべて線形であるかどうかを示します。
- 項項が全体的に対称（複素数の場合で共役）か、少なくとも対称性に影響を与えていないか。2つの異なる変数にで対称に宣言された項は全体線形システム（項と項の転置）で2回追加されます。
- その項が保磁力に影響を与えないか。
- 項が実数のバージョンを持っているかどうか。もしそうなら、`asm_real_tangent_terms` メソッドを再定義する必要があります。
- 項が複素数バージョンを持っているかどうか。もしそうなら、`asm_complex_tangent_terms` メソッドを再定義する必要があります。

`asm_real_tangent_terms` メソッドは、タンジェントシステムの構築のための `model` オブジェクトによって呼び出されます。`model` オブジェクトはブリックにフレームワーク全体を渡してその項を構築します。`asm_real_tangent_terms` メソッドのパラメータ `md` は項を呼び出すモデルであり、`ib` はモデルの項数です。パラメータ `var1` はこのモデルで定義される項で必要とされる変数/データ名の配列です。`mims` は `mesh_im` ポインタの配列です。これは、項を組み立てるために必要な積分方法に対応します。`mat1` は計算された行列の配列です。`vec1` は計算されるベクトルの配列です（`rhs` または残差ベクトル）。`vec1_sym` は対称項に対してのみ計算され、第2の変数の `rhs` に対応するベクトルの配列です。項は任意の数の項を持つことができます。それぞれの項について、少なくとも対応する行列または対応するベクトルを満たされなければなりません（または2つとも、非線形の場合のみ、次のセクションを参照）。`region` は、特定の領域に項を組み立てることを示すメッシュ領域番号です。`n1` は非線形項のみです。これは接線行列または残差または両方を計算するか（線形項の場合は、すべての呼び出しごとに計算されます）を示します。

上で定義した非常に単純なラプラシアン・項の場合、1つの変数のみが使用され、データはなく、1つの項しか存

在しません。行

```
GMM_ASSERT1(matl.size() == 1,
            "My Laplacian brick has one and only one term");
GMM_ASSERT1(mims.size() == 1,
            "My Laplacian brick need one and only one mesh_im");
GMM_ASSERT1(varl.size() == 1 && datal.size() == 0,
            "Wrong number of variables for my Laplacian brick");
```

は必須ではなく、項の数 (1)、積分法 (1)、変数 (1)、データ (0) が `asm_real_tangent_terms` メソッドに渡されるのに問題ない値であることを確認するだけです。

行

```
const getfem::mesh_fem &mf_u = md.mesh_fem_of_variable(varl[0]);
const getfem::mesh_im &mim = *mims[0];
```

はラプラシアン項が追加される変数の `mesh_fem` オブジェクトで `mesh_im` オブジェクトは積分法のリストです。最後に

```
gmm::clear(matl[0]);
getfem::asm_stiffness_matrix_for_homogeneous_laplacian
(matl[0], mim, mf_u, region);
```

はファイル `getfem/getfem_assembling.h` で定義されているラプラシアン項の標準構築プロシージャを呼び出します。 `matl` の行列は適切なサイズであることが保証されていますが、 `asm_real_tangent_terms` の呼び出しの前に初期化されない可能性があるため、初期化メソッドが必要です。

この単純な項は項が 1 つしかなく、線形であることに注意してください。線形ブリックの場合、行列または右辺のベクトルのどちらか一方を埋める必要がありますが、両方を埋める必要はありません。項の宣言に応じて。モデルにブリックを積分する方法は以下を参照してください。

Lagrange 乗数の使用により Dirichlet 条件を規定する単純な項の 2 番目の例を見てみましょう。Dirichlet 条件は、

$$u = u_D \text{ on } \Gamma,$$

ここで、 u は変数です u_D は与えられた値で、 Γ は対象とする領域の境界上にある部分です。Lagrange 乗数で規定されたこの条件に対応する弱形式項は、

$$\int_{\Gamma} u \mu \, d\Gamma = \int_{\Gamma} u_D \mu \, d\Gamma, \forall \mu \in M,$$

ここで、 M は適切な乗数空間です。グローバル線形システムへの寄与は、図 [シンプルな Dirichlet ブリックの寄与](#) で見ることができます。行列 B は変数の有限要素空間 u と乗数の有限要素空間 μ の間の“質量行列”です。 L_u はデータ u_D に対応する右辺です。

項は次のように定義できます。


```

my_Dirichlet_brick(void)
{ set_flags("My Dirichlet brick", true /* linear */,
           true /* symmetric */,
           false /* coercivity */,
           true /* real version defined */,
           false /* no complex version */);
}
};

```

このブリックにも項が 1 つしかありませんが、マトリックス部分と右辺部分の両方が定義されています。Dirichlet 条件が規定されているプライマリ変数と、境界に対応するメッシュ領域に定義される乗数変数 (add_multiplier メソッドでモデルに追加する必要があります) の 2 つの変数が関係しています。項は対称であると宣言されます (次のセクションを参照)。

行

```

const getfem::model_real_plain_vector &A = md.real_variable(datal[ind]);
const getfem::mesh_fem *mf_data = md.pmesh_fem_of_variable(datal[ind]);

```

は Dirichlet 状態の右辺に対応するデータの値とこのデータが定義されている *mesh_fem* にアクセスすることを許可します。データが一定である場合 (fem で記述されていない場合)、mf_data はヌルポインタです。

行

```

if (mf_data)
  getfem::asm_source_term(vecl[0], mim, mf_mult, *mf_data, A, region);
else
  getfem::asm_homogeneous_source_term(vecl[0], mim, mf_mult, A, region);

```

は右辺を構築します。2 つのバージョンは有限要素法または一定サイズのデータで定義されたデータに対応します。

(+ 非線形項を持ついくつかの例...)

22.4 モデルに項を追加する方法

モデルに項を追加するには、モデルに特定の情報を渡す必要があります。

- ブリック自体へのポインタ。
- ブリックの項に関係する一連の変数名。
- データ名のセットは、ブリックの項に関係します。
- 項の説明のリスト。

- 積分方法のリスト。
- 最終的に関連するメッシュ領域。

これは、オブジェクトメソッド *model* の呼び出しによって行われます。

```
md.add_brick(pbr, const getfem::model::varnamelist &varnames,
             const getfem::model::varnamelist &datanames,
             const getfem::model::termlist &terms,
             const getfem::model::mimlist &mims,
             size_t region);
```

このメソッドは、モデル内の項のインデックスを返します。このメソッドの呼び出しは、多くの状況に適応できるため、かなり複雑です。新しいブリックの構築は、このメソッドを呼び出すモデルに新しいブリックを追加し、より簡単に使用できる関数の定義に従わなければなりません。

例えば、上記の単純なラプラシアン項の場合、この関数は次のように定義することができます:

```
size_t add_my_Laplacian_brick(getfem::model &md, const getfem::mesh_im &mim,
                             const std::string &varname,
                             size_t region = size_t(-1)) {
    getfem::pbrick pbr = new my_Laplacian_brick;
    getfem::model::termlist tl;

    tl.push_back(getfem::model::term_description(varname, varname, true));
    return md.add_brick(pbr, getfem::model::varnamelist(1, varname),
                       getfem::model::varnamelist(), tl,
                       getfem::model::mimlist(1, &mim), region);
}
```

この関数はあなたのブリックのユーザーによって呼び出されます。 `getfem::model::varnamelist` 型は `std::vector<std::string>` であり、変数名の配列を表します。 `getfem::model::mimlist` 型は `std::vector<const getfem::mesh_im *>` であり、積分メソッドへのポインタの配列を表します。 `getfem::model::termlist` は項の記述の配列です。項には2種類あります。線形（接線）システムに右辺のみを追加する項は次のようにリストに追加する必要があります。

```
tl.push_back(getfem::model::term_description(varname));
```

そして、線形システムの行列に寄与する項は次のようにリストに追加する必要があります。

```
tl.push_back(getfem::model::term_description(varname1, varname2, true/false));
```

この場合、行列項は変数 “varname1” に対応する行と変数 “varname2” に対応する列に追加されます。第3のパラメータであるブール値はその項が対称であるかどうかを宣言します。対称であり、2つの変数が異なる場合、構築手順は対応する項およびその転置を加えます。項の数は任意です。宣言された各項に対して、項は対応する右辺のベクトル（上の `asm_real_tangent_terms` のパラメータ `vec1`）または/および行列の項

(`asm_real_tangent_terms` のパラメータ `mat1`) を宣言します。非線形項では、行列と右辺ベクトルの両方を満たす必要があることに注意してください。線形項の場合、行列項と宣言された項の右辺が満たされている場合は無視されます。

変数名とデータ名は 2 つの別々の配列で与えられます。これは、どちらの場合でも項の依存関係が同じではないためです。線形項は、データの値が変更された場合には再計算されなければなりません、変数の値が変更された場合は再計算されません。

上記の単純な Dirichlet 項を追加する機能は、次のように定義することができます:

```
size_t add_my_Dirichlet_condition_brick(model &md, const mesh_im &mim,
                                       const std::string &varname,
                                       const std::string &multname,
                                       size_t region,
                                       const std::string &dataname) {
    pbrick pbr = new my_Dirichlet_brick;
    model::termlist tl;
    tl.push_back(model::term_description(multname, varname, true));
    model::varnamelist vl(1, varname);
    vl.push_back(multname);
    model::varnamelist dl;
    if (dataname.size()) dl.push_back(dataname);
    return md.add_brick(pbr, vl, dl, tl, model::mimlist(1, &mim), region);
}
```

再びここでは、この項は対称であると宣言され、次に行列項とその転置が追加されます。

22.5 汎用的な構築ブリック

1 つの変数または複数の変数に項を追加することは、セクション [任意の項を計算する - 高水準の汎用的な構築手順 - 弱形式言語](#) で説明されている弱形式言語を直接使用することです。より汎用的な方法は次の関数を使用することです

```
size_type getfem::add_nonlinear_term(md, mim, expr,
                                     region = -1, is_sym = false, is_coercive = false);
```

これは、メッシュ領域 `region` に構築メソッド `mim` を、構築文字列 `expr` を使用して、モデル `md` に要素を追加します。結果が対称であれば、5 番目の引数で指定し、6 番目の引数で強制的に指定できます。後者の対称性と強制性は、正しい線形ソルバを決定するために使用されます。理解していない場合は、何も示さないことを推奨します。

しかし、このブリックは、式が非線形であるとみなします。このブリックは、特に、いくつかの変数間の非線形結合項を得るために示されています。これは、特に項の構築がモデルの構築の各呼び出しで実行され、Newton 法が

問題を解決するために使用されることを意味します。項が実際に線形の場合は、代わりに:

```
size_type getfem::add_linear_term(md, mim, expr,
    region = -1, is_sym = false, is_coercive = false);
```

同じ議論で。逆に、この要素は常に `expr` に対応する項は線形であると仮定し、使用されたデータが変更されない場合、構築は 1 回だけ実行されます。したがって、あなたの表現は実際には各変数に関して線形（アファイン）であることに注意しなければなりません。それ以外の場合、結果は保証されません。表現のソース項が考慮されません。それでも線形の問題では、次のような理由で単一のソース項を組み立てることは可能です:

```
size_type getfem::add_source_term(md, mim, expr, region = -1);
```

再び対称性と強制性を除いて同じ議論をします。このブリックは、対応する次数 1 項（残差ベクトル）の集合を実行しそれを右辺として問題に追加します。構築は 1 回だけ実行されるため、この項はモデルの変数に依存してはいけません（ただし定数に当てはまる可能性があります）。

例えば、モデルの定義済み変数 `u` で Poisson 問題を解きたい場合は、対応する事前定義された要素を使用するか（下記参照）、単純に:

```
getfem::add_nonlinear_term(md, mim, "Grad_u.Grad_Test_u - F*Test_u", -1, true, true);
```

ここで `F` は右辺を表すモデルのあらかじめ定義された定数です。もちろん、そうすることで、Newton 法が呼び出されます。だから、より適切な方法は次のように線形要素を使用することです:

```
getfem::add_linear_term(md, mim, "Grad_u.Grad_Test_u", -1, true, true);
getfem::add_source_term(md, mim, "F*Test_u");
```

現時点では、複素変数のある問題に対して弱形式言語の使用は不可能であることに注意してください。

22.6 汎用的な楕円ブリック

この要素は、モデルの変数に楕円形の項を追加します。楕円項の形状は、変数と与えられた係数の両方に依存します。これは次の項に対応します:

$$-\operatorname{div}(a\nabla u),$$

ここで、 a は係数であり、 u は変数です。係数は、スカラー、行列またはテンソル 4 次のオーダです。変数はベクトル値でも、そうでなくてもかまいません。これは、ブリックがいくつかの異なる状況を扱うことを意味します。係数がスカラーまたは行列で、変数がベクトル値の場合、その項は成分ごとに加算されます。4 つのテンソル係数はベクトル値の変数にのみ使用できます。係数は一定であっても FEM で記述されてもよいです。もちろん、係数が有限要素法（テンソル場）上に記述されたテンソルである場合、対応するデータは巨大ベクトルとなり得ます。行列/テンソルの成分は、係数（BLAS との適合性）に対応するデータベクトルに Fortran 順序（列方向）で格納しなければなりません。与えられた行列/テンソルの対称性と保磁力は検証されません（しかし仮定されます）。

この要素は、2つの関数のおかげでモデル md に追加することができます。最初のものは:

```
size_type getfem::add_Laplacian_brick(md, mim, varname, region = -1);
```

それは 1 (ラプラシアン項) に等しい定数係数を持つモデルの変数 varname に対して相対的に楕円形の項を加えます。これは、ラプラス演算子に対応します。mim はその項を計算するために使用される積分方法です。region はオプションの領域番号です。省略されている場合、項はメッシュ全体で計算されるものとみなされません。関数の結果は、モデルのブリックのインデックスです。

2番目の関数は:

```
size_type getfem::add_generic_elliptic_brick(md, mim, varname, dataexpr, region = -1);
```

モデル変数にも依存したとしても (例えば “1”, “sin(X[0])” or “Norm(u)” のような) 弱形式言語の正規表現でなければならない式 dataexpr によって与えられた任意の係数が増えられます。(モデルの宣言されたデータでなければならない複素数バージョンを除く)。

この要素では非常に汎用的な方程式が得られることに注意してください。例えば、テンソルデータを用いて線形異方性弾性を得ることができる。テンソル 4 次を使用する場合、対応する弱定式化項は次のようになります

$$\int_{\Omega} \sum_{i,j,k,l} a_{i,j,k,l} \partial_i u_j \partial_k v_l dx$$

ここで、次のようになります。 $a_{i,j,k,l}$ はテンソル 4 次の次数であり、未知数 k の j 成分の $\partial_i u_j$ は次の i^{th} 変数の偏微分です。 v は試行関数です。しかし、線形等方性弾性の場合、より適合した要素が利用可能です (下記参照)。

要素は作業する複素数バージョンを持っています。

22.7 Dirichlet 条件ブリック

Dirichlet 条件ブリックの目的は、モデル変数のために領域の境界の一部に Dirichlet 条件を規定することです。これは、この変数の値が境界で規定されていることを意味します。このブリックは 3 つのバージョンがあります (*Dirichlet* および接触境界条件に対する *Nitsche* 法の節も参照してください)。最初のバージョンは Dirichlet 乗数を定めます。関連する弱形式の項は次のとおりです。

$$\int_{\Gamma} u \mu d\Gamma = \int_{\Gamma} u_D \mu d\Gamma, \forall \mu \in M.$$

ここで、 u は変数です M は乗数の空間です、 u は変数、 Γ は Dirichlet 境界です。このバージョンでは乗数を表すために追加の変数が必要です。これは、モデルに直接行うことも、以下の関数で行うこともできます。乗数で指定された Dirichlet 条件を追加できる 3 つの関数があります。最初のものは:

```
add_Dirichlet_condition_with_multipliers(md, mim, varname,
                                          multname, region,
                                          dataname = std::string());
```

メッシュ領域 *region* (境界でなければならない) 上の乗数変数 *multname* のおかげで、*varname* に **Dirichlet** 条件を追加しました。その境界上の変数の値は、あらかじめモデル内で定義されていなければならないデータ *dataname* によって記述されます。データが省略された場合、**Dirichlet** 条件は均質なもの (境界上の消失変数) とみなされます。データは、FEM 上で一定または記述することができます。また、変数に応じてスカラー値またはベクトル値にすることもできます。変数 *multname* はメソッド `add_multiplier` でモデルに追加する必要があります。この関数は、モデル内のブリックのインデックスを返します。2 番目の機能は:

```
add_Dirichlet_condition_with_multipliers(md, mim, varname,
                                         mf_mult, region,
                                         dataname = std::string());
```

唯一の違いは、*multname* は *mf_mult* に置き換えられます。つまり、乗算器が構築される有限要素のみが与えられます。この関数は、乗数変数をモデルに追加します。3 番目の関数は非常に似ています:

```
add_Dirichlet_condition_with_multipliers(md, mim, varname,
                                         degree, region,
                                         dataname = std::string());
```

パラメータ *mf_mult* は乗数とその程度の古典的な有限要素法に基づいて構築されることを示す整数の *degree* に置き換えられます。

全ての場合において、変数が *model* オブジェクトの `add_multiplier` メソッドによって追加されたとき *mesh_fem* は (`partial_mesh_fem_object` により) 考慮されている境界に寄与していない自由度しか保持しないようにフィルタリングされます。

最後に、乗算器の変数名は関数により:

```
mult_varname_Dirichlet(md, ind_brick);
```

ここで *ind_brick* はモデルのブリックのインデックスです。この関数は別の種類のブリックに適用された場合、未定義の動作をします。

Dirichlet 条件のブリックの第 2 のバージョンは、ペナルティを伴うものです。このブリックを追加できる関数は:

```
add_Dirichlet_condition_with_penalization(md, mim, varname,
                                           penalization_coeff, region,
                                           dataname = std::string(),
                                           *mf_mult = 0);
```

ペナルティは変数の質量行列を計算し、ペナルティ係数を掛けたものに剛性行列を加算することからなります。パラメータ *mf_mult* (`get_fem::mesh_fem` オブジェクトへのポインタ) はオプションです。**Dirichlet** 条件をロックするために、それを弱めることができます。この場合、ペナルティ行列は $B^T B$ のような形式になります。ここで、 B は、変数 *varname* の試行関数と乗数空間ペナルティ係数はモデルのデータとして追加され、次の関数によって変更することができます:

```
change_penalization_coeff(md, ind_brick, penalisation_coeff);
```

Dirichlet 条件ブリックの第 3 バージョンは、線形システム（非線形問題のための接線系）の簡略化を使用します。基本的には、規定された自由度に対応する線の対角成分に 1 を強制し、それに応じていくつかのゼロを有する線を完成させます（対称問題については、いくつかのゼロで列を完成させる）。これは、Dirichlet 条件を処理するための、単純で効率的な方法です。ただし、Dirichlet 条件で規定されている自由度を特定できる場合にのみ適用できます。したがって、有限要素法、Hermite 要素法の使用を慎重に使用しなければならず、ベクトル問題に関する通常の（または一般化された）Dirichlet 条件には適用できません。このブリックを追加できる関数は:

```
add_Dirichlet_condition_with_simplification(md, varname, region,
                                             dataname = std::string());
```

dataname が省略されている場合は、同種の Dirichlet 条件が適用されます。*dataname* が与えられた場合、Dirichlet 条件が適用される変数 *varname* と同じ有限要素法で定数または記述されなければならないという制約があります。加えて、*dataname* が定数の場合、Lagrange 有限要素法にしか適用できません。

22.8 一般化 Dirichlet 状態ブリック

一般化された Dirichlet 条件は、タイプのベクトル場 u の境界条件である

$$Hu = r$$

ここで、 H は行列フィールドです。対応するブリックを追加する関数は、標準的な Dirichlet 条件のものと似ていますが、補足パラメータ *Hname* を必要とします。これは、 H に対応するデータの名前を与えます。このデータは、スカラーの *fem* または定数行列に記述された行列フィールドにすることができます。

```
add_generalized_Dirichlet_condition_with_multipliers(md, mim, varname,
                                                    multname, region,
                                                    dataname, Hname);
```

```
add_generalized_Dirichlet_condition_with_multipliers(md, mim, varname,
                                                    mf_mult, region,
                                                    dataname, Hname);
```

```
add_generalized_Dirichlet_condition_with_multipliers(md, mim, varname,
                                                    degree, region,
                                                    dataname, Hname);
```

```
add_generalized_Dirichlet_condition_with_penalization(md, mim, varname,
                                                       penalization_coeff, region,
                                                       dataname, Hname);
```

22.9 点列制約ブリック

点列制約ブリックは、ブリックのような **Dirichlet** 条件であり、領域の所与の点に未知の値を処方することを可能にします。これらの点は、必ずしもメッシュの頂点である必要はなく、未知数が記述されている有限要素法の自由度に対応する点列である必要はありません。

スカラー場変数については、 N_p 点 $x_i, i = 1 \dots N_p$ の集合が与えられると、ブリックはこれらの点で変数の値を指定すると、

$$u(x_i) = l_i, \quad i = 1 \dots N_p,$$

ここで、 u はスカラーフィールドで、 l_i はポイント x_i に指定する値です。

ベクトルフィールド変数の場合、 N_p 点数 $x_i, i = 1 \dots N_p$ の集合が与えられた場合、ブリックは変数の 1 つのブリックの値をこれらの点で処理することを可能にします、すなわち、条件を強制します。

$$u(x_i) \cdot n_i = l_i, \quad i = 1 \dots N_p,$$

ここで、 n_i は次のようなベクトルです。例えば $u(x_i) \cdot n_i$ は指定する成分を表します。

ブリックには 2 つのバージョンがあります: ペナルティバージョンと乗算機能付きバージョンです。呼び出しは次のとおりです:

```
add_pointwise_constraints_with_penalization(md, varname, penalisation_coeff,
      dataname_pt, dataname_unitv = std::string(),
      dataname_val = std::string());
```

```
add_pointwise_constraints_with_given_multipliers(md, varname, multname,
      dataname_pt, dataname_unitv = std::string(),
      dataname_val = std::string());
```

```
add_pointwise_constraints_with_multipliers(md, varname, dataname_pt,
      dataname_unitv = std::string(), dataname_val = std::string());
```

与えられた乗数固定サイズの変数を持つものと、適切なサイズの乗数変数をモデルに自動的に追加するものとの 2 種類があります。データ `dataname_pt`、`dataname_unitv`、`dataname_val` は最初に `model` に追加する必要があります。`dataname_pt` は、変数 `varname` の値を規定する点の座標を含むベクトルでなければなりません。したがって、次のようなサイズ NN_p です。ここで、 N はメッシュの次元です。`dataname_unitv` はスカラーフィールド変数では無視されます。ベクトルフィールド変数の場合はベクトル n_i を含むはずで、その場合、サイズは次のように QN_p である必要があります。ここで、 Q はベクトルフィールドの次元です。`dataname_val` はオプションで右辺を表します。これは、 l_i という成分を含んでいなければなりません。 l_i のデフォルト値は **0** です。

このブリックは主に、純粋な Neumann 問題の剛性変位を処理するように設計されています。

22.10 ソース項ブリック（および Neumann 条件）

このブリックは、ソース項、すなわちモデルによって線形（接線）システム構築の右側にのみ生じる項を追加します。もし f はソース項の値を表し、そのような項の弱形式は

$$\int_{\Omega} f v \, dx$$

ここで、 v は試行関数です。値 f は、定数であっても有限要素法で記述されてもよい。

また、領域の境界に適用される場合、Neumann 条件を表すこともできます。

モデルにソース項を追加する機能は次のとおりです。

```
add_source_term_brick(md, mim,
                      varname, dataexpr, region = -1,
                      directdataname = std::string());
```

ここで `md` は `model` オブジェクト、`mim` は積分方法、`varname` はソース項が追加されたモデルの変数、`dataexpr` は正規表現でなければなりません弱形式言語の表現（モデルの宣言されたデータでなければならない複素数バージョンを除く）。変数がスカラー値またはベクトル値であるという事実に応じて、スカラー値またはベクトル値でなければなりません。`region` は項が付加されたメッシュ領域です。領域が境界に対応する場合、ソース項は Neumann 条件を表します。`directdataname` はオプションの追加データで、組み立てなくても右側に直接追加されます。

要素は作業用の複素数バージョンを持っています。

わずかに異なる要素は、特に Neumann 条件を扱うために用意されており、次の関数によって追加されています：

```
add_normal_source_term_brick(md, mim,
                              varname, dataexpr, region);
```

基本ソース項ブリックとの違いは、データがベクトルフィールド（変数 `varname` 自体がベクトル値であれば行列フィールド）でなければならず、外側の単位法線を持つスカラー積がそこで実行されるということです。

22.11 あらかじめ定義されたソルバー

いくつかの問題に対しては特定のソルバーを作成する方が便利ですが、モデルを素早くテストするためのソルバーがあります。また、これは独自のソルバを作成するための例としても使用できます。これは `src/getfem/getfem_model_solvers.h` と `src/getfem_model_solvers.cc` で定義され、呼び出しは：

```
getfem::standard_solve(md, iter);
```

ここで `md` は `model` オブジェクトで、`iter` は `Gmm++` の反復オブジェクトです。使用例については、次のセクションも参照してください。

「小さな」問題ではデフォルトの線形ソルバーとして `SuperLU` が使用されることに注意してください。また、`MUMPS` と `GetFEM++` (セクション [線形代数プロセス](#) を参照) をリンクさせ並列バージョンを使用することができます。非線形問題に対しては、Newton 法 (Newton-Raphson 法とも呼ばれる) が用いられます。

また、変数のサブセットに関してのみ問題を解決するために、いくつかの変数 (`model` オブジェクトのメソッド `md.disable_variable(varname)` を使用) を無効にすることができます (無効な変数はデータと見なされます)。例えばグローバル Newton 法を固定点 1 で置き換えます。

`iter` オブジェクトの標準的な初期化は次のとおりであることを思い出してください (`gmm-iter` に関する `Gmm++` のドキュメントを参照) :

```
gmm::iteration iter(1E-7, 1, 200);
```

ここで、`1E-7` は停止基準の相対許容誤差であり、`1` はノイズの多いオプションであり、`200` は最大反復回数です。Newton 法の停止基準は、以下のように構築されます。相対許容誤差の場合 ε 、アルゴリズムは次のときに停止します。

$$\min (\|F(u)\|_1 / \max(L, 10^{-25}), \|h\|_1 / \max(\|u\|_1, 10^{-25})) < \varepsilon$$

$F(u)$ は残差ベクトルです。 $\|\cdot\|_1$ は \mathbb{R}^n 内の古典的な 1-ノルムです、 h は Newton 法によって与えられた探索方向です。 L は推定された (ソース項および Dirichlet ブリックから来る) 外部負荷のノルムであり、 u は探索された変数の現在の状態です。最大値 10^{-25} は L と/または u の値がない異常な場合を避けるためです。

22.12 Poisson 問題の完全な例

次の例は、テストプログラム `tests/laplacian_with_bricks.cc` の一部です。メッシュおよび有限要素法の構成は省略されています。メッシュが構築され、このメッシュに `mf_u` と `mf_rhs` という 2 つの有限要素法が構築されていると仮定します。 `NEUMANN_BOUNDARY_NUM` と `DIRICHLET_BOUNDARY_NUM` は、そのメッシュ上の 2 つの有効な境界指標であるとも仮定されています。コードは、ソース項、Neumann 条件、Dirichlet 条件のデータを構築するために、`mf_rhs` で補間される 3 つの関数の定義から始まります。 `model` オブジェクトの宣言、要素の追加、および問題の解の計算は次の通りです:

```
using bgeot::base_small_vector;
// Exact solution. Allows an interpolation for the Dirichlet condition.
scalar_type sol_u(const base_node &x) { return sin(x[0]+x[1]); }
// Right hand side. Allows an interpolation for the source term.
scalar_type sol_f(const base_node &x) { return 2*sin(x[0]+x[1]); }
```



```

// Gradient of the solution. Allows an interpolation for the Neumann term.
base_small_vector sol_grad(const base_node &x)
{ return base_small_vector(cos(x[0]+x[1]), cos(x[0]+x[1])); }

int main(void) {

    // ... definition of a mesh
    // ... definition of a finite element method mf_u
    // ... definition of a finite element method mf_rhs
    // ... definition of an integration method mim
    // ... definition of boundaries NEUMANN_BOUNDARY_NUM
    // ... and DIRICHLET_BOUNDARY_NUM

    // Model object
    getfem::model laplacian_model;

    // Main unknown of the problem
    laplacian_model.add_fem_variable("u", mf_u);

    // Laplacian term on u.
    getfem::add_Laplacian_brick(laplacian_model, mim, "u");

    // Volumic source term.
    std::vector<scalar_type> F(mf_rhs.nb_dof());
    getfem::interpolation_function(mf_rhs, F, sol_f);
    laplacian_model.add_initialized_fem_data("VolumicData", mf_rhs, F);
    getfem::add_source_term_brick(laplacian_model, mim, "u", "VolumicData");

    // Neumann condition.
    gmm::resize(F, mf_rhs.nb_dof()*N);
    getfem::interpolation_function(mf_rhs, F, sol_grad);
    laplacian_model.add_initialized_fem_data("NeumannData", mf_rhs, F);
    getfem::add_normal_source_term_brick
    (laplacian_model, mim, "u", "NeumannData", NEUMANN_BOUNDARY_NUM);

    // Dirichlet condition.
    gmm::resize(F, mf_rhs.nb_dof());
    getfem::interpolation_function(mf_rhs, F, sol_u);
    laplacian_model.add_initialized_fem_data("DirichletData", mf_rhs, F);
    getfem::add_Dirichlet_condition_with_multipliers
    (laplacian_model, mim, "u", mf_u, DIRICHLET_BOUNDARY_NUM, "DirichletData");

    gmm::iteration iter(residual, 1, 40000);
    getfem::standard_solve(laplacian_model, iter);

    std::vector<scalar_type> U(mf_u.nb_dof());
    gmm::copy(laplacian_model.real_variable("u"), U);

```

```
// ... doing something with the solution ...

return 0;
}
```

ブリックは任意の回数で追加できます。

22.13 Dirichlet および接触境界条件に対する Nitsche 法

GetFEM++ は Dirichlet タイプまたは摩擦境界条件との接触を、Lagrange 乗数を使用せずに弱形式で考慮に入れることができる Nitsche の方法の汎用的な実装を与えます。この方法は、Dirichlet 境界条件を Neumann 境界条件と同様に弱形式項に変換するため、非常に魅力的です。しかしながら、この利点は、対応する Neumann 項の近似を必要とするため、Nitsche 法の実施がモデル依存であることがコストです。Nitsche の方法で境界条件をモデルの変数に追加するには、対応するブリックが、この変数に適用されるすべての偏微分項の Neumann 項の近似にアクセスする必要があります。以下では、変数 u を考慮して、

$$G$$

この変数上のすべての Neumann 項の和。Neumann 項 G は変数 u に依存することが多いことに注意してください。しかしモデルの他の変数にも依存します。これは、例えば、非圧縮弾性の混合式の場合です。Neumann 項は、モデルのいくつかのパラメータ（弾力係数...）にも頻繁に依存しますが、これはその式に含まれると仮定されます。

例えば、ラプラス項（ Δu ）が変数 u に適用される場合、Neumann 項は $G = \frac{\partial u}{\partial n}$ のようになります。ここで、 n は考慮されている境界上の外側の単位法線です。もし u が変形可能な物体の変位を表すなら、Neumann 項は $G = \sigma(u)n$ のようになります。もちろん、その場合 G はいくつかの物性値に依存します。さらに、圧力を表す変数 p を混合した非圧縮ブリックに加えると、 u 上の Neumann 項は次のように p に依存します。 $G = \sigma(u)n - pn$

Nitsche の方法を課しているブリックが、関連する変数に適用されるすべての偏微分項に対して機能する汎用的な実装を可能にするために、モデルに部分微分項を追加する各ブリックは、構築文字列（弱形式言語）が必要です。

これらの式は、`model` オブジェクトの特別なメソッド:

```
expr = md.Neumann_term(variable, region)
```

すべての部分微分項要素が与える式を走査し、適切な操作を実行することによって、すべての Neumann 項の和に対する式を自動的に導出することを可能にします。もちろん、このメソッドを呼び出す前に、すべてのボリューム要素をモデルに追加する必要があります。Neumann 項の導出は、2次偏微分方程式に対してのみ働く。高次 pde の汎用的な実装はより複雑になります。

22.13.1 Dirichlet 条件のための汎用的な Nitsche 法

変数 u が考慮され、条件を規定したいとします

$$Hu = g$$

部分 Γ_D 考慮する領域の境界です。ここで、 H は、スカラーの場合、1 と等しいと見なされるか、ベクトルの場合の恒等行列か、固有値として 1 または 0 のみを持つ特異行列のいずれかです。これにより、ここでは、 u の正接または正接成分のみを処理することができます。たとえば、正規成分のみを処理したい場合、 H は以下のように選択されます nn^T ここで、 n は Γ_D 上の外向きの単位法線です。

この Dirichlet 条件を処理するための Nitsche 法では、問題の弱形式に次の項を加えます

$$\int_{\Gamma_D} \frac{1}{\gamma} (Hu - g - \gamma HG) \cdot (Hv) - \theta (Hu - g) \cdot (HD_u G[v]) d\Gamma,$$

γ と θ は Nitsche 法の 2 つのパラメータです。 v は u に対応する試行関数です。パラメータ θ は正または負が選択できます。 $\theta = 1$ は標準的な状況で対称的な接線項につながるより標準的な方法に対応します $\theta = 0$ は数が減少するという利点を持つ非対称的な方法に対応します。非線形の場合には G の 2 次導関数を必要とせず、 $\theta = -1$ は少なくとも標準的な状況では非平衡保磁力を保証する一種のスキュー対称法である（これは、 γ ）。パラメータ γ は一種のペナルティ・パラメータですが（メソッドは一貫していますが）、次のようになります $\gamma = \gamma_0 h_T$ 、ここで、 γ_0 はメッシュで h_T は要素 T の直径です。 $\theta = -1$ を除く標準的な状況では、パラメータ γ_0 は Nitsche 法の収束を確実にするために十分小さくする必要がありますことに注意してください。

Nitsche 法で Dirichlet 条件をモデルに追加した要素は次のとおりです:

```
getfem::add_Dirichlet_condition_with_Nitsche_method
(model &md, const mesh_im &mim, const std::string &varname,
 const std::string &Neumannterm,
 const std::string &gamma0name, size_type region,
 scalar_type theta = scalar_type(1),
 const std::string &dataname = std::string());
```

この関数は変数 $varname$ とメッシュ領域 $region$ に Dirichlet 条件を追加します。この領域は境界でなければなりません。 $Neumannterm$ は弱形式言語の表現として記述された Neumann 項（Green 式によって得られる）の表現です。この項は、すべてのボリューム要素がモデルに追加されると $md.Neumann_term(varname, region)$ で取得できます。Dirichlet 条件は Nitsche 法で規定されています。 $dataname$ は Dirichlet 条件のオプションの右辺です。これは一定か、または fem 上に記述することができます; Dirichlet 条件が規定されている変数に応じて、スカラー値またはベクトル値を返します。 $gamma0name$ は Nitsche 法のパラメータです。 $theta$ は正または負のスカラー値です。 $theta = 1$ は標準的な対称的なメソッドに対応しています。これは、 $gamma0$ が小さい場合に強制的に行われます。 $theta = -1$ は非強制的強制的なスキュー対称方法に対応します。 $theta = 0$ は Neumann 項の 2 次導関数が非線形問題であっても必要でない最も単純な方法です。モデル内の要素のインデックスを返します。

```
getfem::add_normal_Dirichlet_condition_with_Nitsche_method
(model &md, const mesh_im &mim, const std::string &varname,
 const std::string &Neumannterm,
 const std::string &gamma0name, size_type region,
 scalar_type theta = scalar_type(1),
 const std::string &dataname = std::string());
```

この関数は、ベクトル（またはテンソル）値変数 *varname* とメッシュ領域 *region* の正規成分に Dirichlet 条件を追加します。この領域は境界でなければなりません。 *Neumannterm* は弱形式言語の表現として記述された Neumann 項（Green 式によって得られる）の表現です。この項は、すべてのボリウム要素がモデルに追加されると、`md.Neumann_term(varname, region)` で取得できます。Dirichlet の状態は Nitsche の方法で規定されています。 *dataname* はオプションの Dirichlet 条件の右辺です。これは一定であるか、または fem 上に記述することができます。 *gamma0name* は Nitsche 法のパラメータです。 *theta* は正または負のスカラー値です。 *theta = 1* は標準的な対称メソッドに対応しています。これは、 *gamma0* が小さい場合に強制的に行われます。 *theta = -1* は非強制的強制的なスキュー対称法に対応します。 *theta = 0* は Neumann 項の 2 次導関数が非線形問題であっても必要でない最も単純な方法です。モデル内の要素のインデックスを返します。（この要素は完全にテストされていません）：

```
getfem::add_generalized_Dirichlet_condition_with_Nitsche_method
(model &md, const mesh_im &mim, const std::string &varname,
 const std::string &Neumannterm,
 const std::string &gamma0name, size_type region, scalar_type theta,
 const std::string &dataname, const std::string &Hname);
```

この関数は変数 *varname* とメッシュ領域 *region* に Dirichlet 条件を追加します。このバージョンはベクトルフィールド用です。それは条件 $Hu = r$ を規定します、ここで H は行列のフィールドです。領域は境界でなければなりません。この領域は境界でなければなりません。 *Neumannterm* は弱形式言語の式として記述された Neumann 項（Green 式によって得られる）の表現です。この項は、すべてのボリウム要素がモデルに追加されると、`md.Neumann_term(varname, region)` で取得できます。Dirichlet 条件は Nitsche の方法で規定されています。注意：行列 H はすべての固有値が 1 または 0 に等しくなければなりません。 *dataname* は Dirichlet 条件のオプションの右辺です。それは一定であるか、または有限要素法上に記述することができます。 *gamma0name* は Nitsche のメソッドパラメータです。 *theta* は正または負のスカラー値です。 *theta = 1* は標準的な対称的なメソッドに対応しています。これは、 *gamma0* が小さい場合に強制的に行われます。 *theta = -1* は非強制的強制的な skew 対称方法に対応します。 *theta = 0* は Neumann 項の 2 次導関数が非線形問題であっても必要でない最も単純な方法です。 *Hname* はマトリックスフィールド H に対応するデータです。それは一定の行列でなければならないか、またはスカラーの fem で記述されなければならない。モデル内の要素のインデックスを返します。（この要素は完全にテストされていません）

22.13.2 摩擦条件との接触のための汎用的な Nitsche の方法

ここでは、小さな変形フレームワークで Coulomb 摩擦条件との接触を規定する Nitsche の方法の使用について説明します。これは弱積分接触条件に相当し、対応する節で Lagrange 乗数を使用するものとある程度類似しているため、[弱積分接触条件](#) を参照してください。

表記法を簡略化するためには、 $P_{n,\mathcal{F}}$ を使用してください。次のマップはいくつかの予測に対応しています：

$$P_{n,\mathcal{F}}(x) = -(x \cdot n)_- n + P_{B(0,\mathcal{F}(x \cdot n)_-)}(x - (x \cdot n)n)$$

このアプリケーションは、 \mathbb{R}_- と中心のボールの接線部分 0 と半径 $\mathcal{F}(x \cdot n)_-$ に x の通常部分の投影を作成します。ここで、 \mathcal{F} は摩擦係数です。

これを使って、摺動速度を $\alpha(u_T - w_T)$ のように近似することを考えてみましょう。ここで、 α と w_T は使用されている時間積分スキーム（[弱積分接触条件](#) 参照）に依存して、Nitsche の摩擦接触の項は以下のようになります：

$$\begin{aligned} & - \int_{\Gamma_C} \theta \gamma G \cdot D_u G[v] d\Gamma \\ & + \int_{\Gamma_C} \gamma P_{n,\mathcal{F}} \left(G - \frac{Au}{\gamma} + \frac{gap}{\gamma} n + \frac{\alpha w_T}{\gamma} \right) \cdot \left(\theta D_u G[v] - \frac{v}{\gamma} \right) d\Gamma. \end{aligned}$$

ここで、 Γ_C は接触境界です。 G は Neumann 項でここでは接触境界での応力 σn を表します。そして A は $d \times d$ マトリックスです。

$$A = \alpha I_d + (1 - \alpha) n n^T$$

$\theta = 0$ の値では、大部分の項は消えることに注意してください。

以下の関数は、変数 `varname_u` とメッシュ境界 `region` に Coulomb 摩擦の有無にかかわらず接触条件を追加します。`Neumannterm` は弱形式言語の表現として記述された Neumann 項（Green 式によって得られる）の表現です。この項は、すべてのボリウム要素がモデルに追加されると `md.Neumann_term(varname, region)` で取得できます。接触条件は Nitsche の方法で規定されています。剛性の障害物はデータ `dataname_obstacle` が障害物までの符号付き距離である（有限要素法で補間された）データで記述しなければなりません。`gamma0name` は Nitsche のメソッドパラメータです。`theta` は正または負のスカラー値です。`theta = 1` は標準的な対称的なメソッドに対応しています。これは、`gamma0` が小さい場合に強制的に行われます。`theta = -1` は非強制的強制的なスキュー対称方法に対応します。`theta = 0` は Neumann 項の 2 次導関数が必要でない最も単純な方法です。オプションのパラメータ `dataexpr_friction_coeff` は弱形式言語の任意の表現である摩擦係数です。モデル内の要素のインデックスを返します。

```
getfem::add_Nitsche_contact_with_rigid_obstacle_brick
(model &md, const mesh_im &mim, const std::string &varname_u,
 const std::string &Neumannterm,
```

```
const std::string &expr_obs, const std::string &dataname_gamma0,  
scalar_type theta_,  
std::string dataexpr_friction_coeff,  
const std::string &dataname_alpha,  
const std::string &dataname_wt,  
size_type region);
```

22.14 拘束ブリック

拘束ブリックでは変数に明示的な拘束を追加できます。明示的とは、積分が行われていないことを意味します。 U が変数なら型の拘束です

$$BU = L,$$

次の2つの関数で追加することができます:

```
indbrick = getfem::add_constraint_with_penalization(md, varname,  
                                                    penalisation_coeff, B, L);  
indbrick = getfem::add_constraint_with_multipliers(md, varname,  
                                                    multname, B, L);
```

2番目のケースでは、乗数として機能する（固定サイズの）変数を最初にモデルに追加する必要があります。

ペナルティバージョンでは、 B はプレーンな行を含んではいけません。さもなければ、接線行列全体がプレーンになります。ペナルティ・パラメータは次の関数によって変更することができます:

```
change_penalization_coeff(md, ind_brick, penalisation_coeff);
```

次の2つの関数により、いつでも拘束を変更することができます:

```
getfem::set_private_data_matrix(md, indbrick, B)  
getfem::set_private_data_rhs(md, indbrick, L)
```

ここで `indbrick` はモデル内の要素のインデックスです。

22.15 他の“陽な”要素

2つの（非常に単純な）要素は、接線系に陽な項を加えることを可能にします。

関数:

```
indbrick = getfem::add_explicit_matrix(md, varname1, varname2, B  
                                       issymmetric = false,  
                                       iscoercive = false);
```

変数 `varname1` と `varname2` に対して、接線系に行列 `B` を追加するだけの要素を追加します。与えられた行列は `varname1` の次元数と `varname2` の次元数だけの行を持ちます。2つの変数が異なっていて、`issymmetric` が `true` に設定されている場合は、行列の転置も接線システムに追加されます（デフォルトは `false`）。項が接線系の保磁力に影響を与えない場合、`iscoercive` を `true` に設定します（デフォルトは `false`）。行列は次のコマンドで変更できます:

```
getfem::set_private_data_matrix(md, indbrick, B);
```

関数:

```
getfem::add_explicit_rhs(md, varname, L);
```

変数 `varname` に対して接線系の右辺にベクトル `L` を追加する要素を追加します。与えられたベクトルは変数 `varname` と同じ大きさでなければなりません。ベクトルの値は、コマンドによって変更することができます:

```
getfem::set_private_data_rhs(md, indbrick, L);
```

22.16 Helmholtz 要素

この要素は複素数と実数の Helmholtz 問題を表しています。

$$\Delta u + k^2 u = \dots$$

ここで波数 k は実数または複素数値です。複素数のバージョンでは、複素数モデルを使用する必要があります (`tests/helmholtz.cc` を参照)。

Helmholtz 要素をモデルに追加する機能は:

```
getfem::add_Helmholtz_brick(md, mim, varname, dataexpr, region);
```

`varname` は Helmholtz 項が追加された変数で `dataexpr` は波数です。

22.17 Fourier-Robin 要素

この要素は Fourier-Robin タイプの境界条件を追加するために使用できます:

$$\frac{\partial u}{\partial \nu} = Qu$$

または、スカラー問題の場合

$$\sigma \cdot \nu = Qu$$

線形弾性問題のために。 Q はスカラーの場合のスカラーフィールドまたはベクトルの場合のマトリックスフィールドです。この要素は、スカラーまたはベクトル問題の実数または複素数の両方に使用できます。

このブリックをモデルに追加する関数は

```
add_Fourier_Robin_brick(md, mim, varname, dataexpr, region);
```

ここで `dataexpr` は係数を表すモデルのデータです Q 。これは、弱形式言語の任意の有効な式（モデルのデータでなければならない複素数バージョンを除く）です。

追加の右側にソース項ブリックを追加することができます。

22.18 等方性線形弾性ブロック

この要素は項

$$-div(\sigma) = \dots$$

と

$$\begin{aligned}\sigma &= \lambda \text{tr}(\varepsilon(u))I + 2\mu\varepsilon(u) \\ \varepsilon(u) &= (\nabla u + \nabla u^T)/2\end{aligned}$$

を表します。 $\varepsilon(u)$ は小さなひずみテンソルです。 σ は応力テンソルです。 λ と μ は Lamé 係数です。これは線形等方性弾性の系を表します。ストークス問題を構築するために線形非圧縮要素と一緒に $\lambda = 0$ で使うこともできます。

Lamé 係数と Young 係数 E と Poisson 比 ν は

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)},$$

平面応力近似（2D モデル）を除き、

$$\lambda^* = \frac{E\nu}{(1-\nu^2)}, \quad \mu = \frac{E}{2(1+\nu)},$$

この要素をモデルに追加し Lamé 係数は:

```
ind_brick = getfem::add_isotropic_linearized_elasticity_brick
    (md, mim, varname, data_lambda, data_mu,
     region = size_type(-1));
```

ここで、`dataname_lambda` と `dataname_mu` は、Lamé 係数を表すモデルのデータです。

このブリックをモデルに追加し、Young 率と Poisson 比でパラメータ化した関数は:

```
ind_brick = getfem::add_isotropic_linearized_elasticity_brick_pstrain
    (md, mim, varname, data_E, data_nu, region = size_type(-1));
```

この要素は、2D メッシュ（および 3D メッシュ上の標準モデル）に適用されるときに平面歪み近似を表します。2D メッシュの平面応力近似を得るために、


```
ind_brick = getfem::add_isotropic_linearized_elasticity_brick_pstress
            (md, mim, varname, data_E, data_nu, region = size_type(-1));
```

3D メッシュの場合、前の 2 つの要素で同じ結果が得られます。

関数:

```
getfem::compute_isotropic_linearized_Von_Mises_or_Tresca
    (md, varname, dataname_lambda, dataname_mu, mf_vm, VM, tresca_flag = false);
```

varname に格納されている変位フィールド上の Von Mises 基準 (tresca_flag が true に設定されている場合は Tresca) を計算します。応力は *mesh_fem* mf_vm を呼び出して VM というベクトルに格納します。2D 平面応力近似には有効ではなく、Lamé 係数でパラメータ化されています。関数は:

```
getfem::compute_isotropic_linearized_Von_Mises
    (md, varname, data_E, data_nu, mf_vm, VM);
```

```
getfem::compute_isotropic_linearized_Von_Mises
    (md, varname, data_E, data_nu, mf_vm, VM);
```

Young 率と Poisson 比でパラメータ化された Von Mises 応力を計算し、2 番目の方程式が 2D メッシュに適用されたときに 2D 平面応力近似は有効です (2 つの関数は 3D 問題に対して同じ結果を与えます)。

プログラム tests/elastostatic.cc は線形化された等方性弾性要素の使用モデルとして取ることができます。

22.19 線形非圧縮性 (またはほぼ非圧縮性) ブリック

このブリックはタイプの問題で線形非圧縮性条件 (またはほぼ非圧縮性の条件) を追加します。

$$\operatorname{div}(u) = 0, \quad (\text{or } \operatorname{div}(u) = \varepsilon p)$$

この制約は混合式で導入された圧力を表す Lagrange 乗数によって強制されます。

この非圧縮性条件を追加する関数は:

```
ind_brick = getfem::add_linear_incompressibility
            (md, mim, varname, multname_pressure, region = size_type(-1),
             dataexpr_penal_coeff = std::string());
```

ここで varname は非圧縮性条件が規定されている変数であり、multname_pressure は乗数 (圧力) を表すスカラー fem で記述される変数であり、dataexpr_penal_coeff はほぼ非圧縮状態のオプションのペナルティ係数です。

ほぼ非圧縮性の均質線形弾性では、次のようになります。 $\varepsilon = 1/\lambda$ ここで、 λ は Lamé 係数の 1 つであり、 ε はペナルティ係数です。

例えば、次のプログラムは、境界項 0 上のソース項と均質な Dirichlet 条件を持つストークス問題を定義しています。mf_u、mf_data、mf_p は同じメッシュ上の有効な有限要素法でなければなりません。mim は同じメッシュ上で有効な積分メソッドでなければなりません:

```
typedef std::vector<getfem::scalar_type> plain_vector;
size_type N = mf_u.linked_mesh().dim();

getfem::model Stokes_model;

laplacian_model.add_fem_variable("u", mf_u);

getfem::scalar_type mu = 1.0;
Stokes_model.add_initialized_data("lambda", plain_vector(1, 0.0));
Stokes_model.add_initialized_data("mu", plain_vector(1, mu));

getfem::add_isotropic_linearized_elasticity_brick(Stokes_model, mim,
                                                  "u", "lambda", "mu");

laplacian_model.add_fem_variable("p", mf_p);
getfem::add_linear_incompressibility(Stokes_model, mim, "u", "p");

plain_vector F(mf_data.nb_dof()*N);
for (int i = 0; i < mf_data.nb_dof()*N; ++i) F(i) = ...;
Stokes_model.add_initialized_fem_data("VolumicData", mf_data, F);
getfem::add_source_term_brick(Stokes_model, mim, "u", "VolumicData");

getfem::add_Dirichlet_condition_with_multipliers(Stokes_model, mim,
                                                  "u", mf_u, 1);

gmm::iteration iter(residual, 1, 40000);
getfem::standard_solve(Stokes_model, iter);

plain_vector U(mf_u.nb_dof());
gmm::copy(Stokes_model.real_variable("u"), U);
```

ほぼ非圧縮性の条件の例は、プログラム tests/elastostatic.cc にあります。

22.20 質量ブリック

この要素は弱形式項

$$\int_{\Omega} \rho u \cdot v \, dx + \dots$$

を表します。これは、主に静的な問題の質量項を表しますが、他の用途にも使用できます（境界で使用できます）。基本的に、この要素はある変数に関して接線系上に質量行列を追加します。

この要素をモデルに追加する関数は:

```
ind_brick = getfem::add_mass_brick
            (md, mim, varname, dataexpr_rho="", region = size_type(-1));
```

ここで `dataexpr_rho` は密度 ρ を表すオプションの式です。これを省略すると、密度は 1 と仮定されます。

22.21 Bilaplacian と Kirchhoff-Love プレートブリック

以下の関数:

```
ind = add_bilaplacian_brick(md, mim, varname, dataname,
                           region = size_type(-1));
```

は変数 `varname` とメッシュ領域 `region` に `bilaplacian` ブリックを追加します。これは、項 $\Delta(D\Delta u)$ を表します。ここで、 $D(x)$ は `dataname` によって決定される係数であり、定数または f.e.m 上で記述されます。対応する弱形式は $\int D(x)\Delta u(x)\Delta v(x)dx$ です。

Kirchhoff-Love プレートモデルの場合、弱形式は少し異なります（前の形よりも安定しています）。その項を追加する関数は:

```
ind = add_bilaplacian_brick_KL(md, mim, varname, dataname1, dataname2,
                               region = size_type(-1));
```

これは変数 `varname` とメッシュ領域 `region` に `Bilaplacian` 要素を追加します。これは $\Delta(D\Delta u)$ のような項を表します。ここで、 $D(x)$ は `dataname1` によって決定される屈曲率です。この項は、Poisson 比 `dataname2` を有する Kirchhoff-Love プレートモデルに続く部分によって積分される。

4 次の偏微分方程式に適切な境界条件を追加するための特定の要素があります。最初のものは:

```
ind = add_normal_derivative_source_term_brick(md, mim, varname,
                                               dataname, region);
```

これは変数 `varname` とメッシュ領域 `region` に正のソース項ブリック $F = \int b \cdot \partial_n v$ を追加します。これは線形システムの右側を更新します。`dataname` は b を表し、`varname` は v を表します。

Neumann 項は以下の要素により追加することができます:

```
ind = add_Kirchhoff_Love_Neumann_term_brick(md, mim, varname,
      dataname1, dataname2, region);
```

変数 *varname* とメッシュ領域 *region* に Kirchhoff-Love モデルの Neumann 項ブリックを追加しました。*dataname1* は曲げモーメントテンソルを表し *dataname2* はその逆テンソルを表します。

そして、次の要素で正規 Dirichlet の Dirichlet 条件を規定することができます:

```
ind = add_normal_derivative_Dirichlet_condition_with_multipliers
      (md, mim, varname, multname, region, dataname = std::string(),
      R_must_be_derivated = false);
```

```
ind = add_normal_derivative_Dirichlet_condition_with_multipliers
      (md, mim, varname, mf_mult, region, dataname = std::string(),
      R_must_be_derivated = false);
```

```
ind = add_normal_derivative_Dirichlet_condition_with_multipliers
      (md, mim, varname, degree, region, dataname = std::string(),
      R_must_be_derivated = false);
```

これらの要素は、変数 *varname* の標準偏差とメッシュ領域 *region* (これは境界でなければなりません) に Dirichlet 条件を追加します。汎用的な形式は $\int \partial_n u(x)v(x) = \int r(x)v(x)\forall v$ のとおりです。ここで、 $r(x)$ は Dirichlet 条件 (均質条件の場合は 0) の右辺で、 v は変数 *multname* によって定義された乗数の空間 (第 1 のバージョン)、または *mf_mult* で定義された有限要素法 (第 2 のバージョン)、または単 *region* により定義された境界の部分の次数による Lagrange 有限要素法 (第 3 のバージョン) です。*dataname* は、Dirichlet 条件の右辺を表すオプションのパラメータです。*R_must_be_derivated* が *true* に設定されている場合、*dataname* の標準偏差が考慮されます。

テストプログラム `bilaplacian.cc` はこの要素の使用例です。

22.22 Mindlin-Reissner プレートモデル

このブリックは等方性プレート用の古典的な Mindlin-Reissner 曲げモデルを実装しています。

22.22.1 Mindlin-Reissner プレートモデル

$\Omega \subset \mathbb{R}^2$ は厚さ ϵ のプレートのミッドプレーンの参照構成です

等方性材料の Mindlin-Reissner モデルの弱定式化は以下のように書くことができます。ここで u_3 は横方向の変位

であり、 θ は中間平面に垂直なファイバーの回転です:

$$\int_{\Omega} D\epsilon^3 ((1-\nu)\gamma(\theta) : \gamma(\psi) + \nu \operatorname{div}(\theta) \operatorname{div}(\psi)) dx + \int_{\Omega} G\epsilon(\nabla u_3 - \theta) \cdot (\nabla v_3 - \psi) dx = \int_{\Omega} F_3 v_3 + M \cdot \psi dx,$$

すべての許容可能な試行関数について $v_3 : \Omega \rightarrow \mathbb{R}$, $\psi : \Omega \rightarrow \mathbb{R}^2$ であり:

$$D = \frac{E}{12(1-\nu^2)}, \quad G = \frac{E\kappa}{2(1+\nu)},$$

$$\gamma(\theta) = (\nabla\theta + \nabla\theta^T)/2,$$

$$F_3 = \int_{-\epsilon/2}^{\epsilon/2} f_3 dx_3 + g_3^+ + g_3^-,$$

$$M_\alpha = \epsilon(g_\alpha^+ - g_\alpha^-)/2 + \int_{-\epsilon/2}^{\epsilon/2} x_3 f_\alpha dx_3, \alpha \in \{1, 2\},$$

f は 3次元板の内部に加えらるる体積力であり、 g^+ と g^- は板の上面と底面に加えらるる力です。 E は Young 率、 ν Poisson 比そして κ はせん断補正係数 (通常 5/6 に設定) です。

古典的な境界条件は次のとおりです。

- 単純支持: u_3 上の Dirichlet 条件。
- 固定支持: u_3 と θ の Dirichlet 条件。
- 既定の横力: u_3 上の境界ソース項。
- 既定のモーメント: $theta$ 上の境界源項。

このモデルの重要な問題は、直接的 Galerkin 法が満足できる近似を与えないため、いわゆるせん断ロッキングが発生することです。せん断ロッキングの問題を回避するには、積分の低減・横方向のせん断項の投影・混合方法があります。2つの高速な方法が提案されています。

横断せん断項の積分を減少させる

この方法では単に下位の積分方法を使用して数値的に計算します

$$\int_{\Omega} G\epsilon(\nabla u_3 - \theta) \cdot (\nabla v_3 - \psi) dx$$

この方法は少なくとも回転変位と横変位の両方を 1 階微分積分法を用いた Q1 四辺形要素 (いわゆる QUAD4 要素) で近似するとき適切に機能します。

横断方向せん断項の射影

もう 1 つの方法はいくつかの混合方法の投影の再解釈に対応する MITC 要素（テンソリアル成分の混合補間）に由来します。このタイプの最も汎用的な要素は回転度の低い Raviart-Thomas 要素（RT0）の横断方向せん断項の投影を伴う四辺形要素 Q1 に対応する MITC4 です（[ba-dv1985], [br-ba-fo1989] を参照）。これは横せん断項が

$$\int_{\Omega} G\epsilon P^h(\nabla u_3 - \theta) \cdot P^h(\nabla v_3 - \psi) dx$$

ここで、 $P^h(T)$ は回転された RT0 空間上の要素の大きさ L^2 -投影です。現時点では、実装されている唯一のプロジェクトションは、前に説明したものです（四角形要素の回転 RT0 空間上の投影）。より高度の要素と 3 角形の要素はライブラリで見つけることができます（[Mi-Zh2002], [br-ba-fo1989], [Duan2014] を参照）。また、以下のことに注意してください。 $P^h(\nabla u_3) = \nabla u_3$ であるため、項は

$$\int_{\Omega} G\epsilon(\nabla u_3 - P^h(\theta)) \cdot (\nabla v_3 - P^h(\psi)) dx$$

に減らすことができることに留意してください。弱形式言語の記述であれば基本的な要素投影の定義の原則が説明されています（[初等変換](#) を参照）。また例はファイル `src/getfem_linearized_plates.cc` にあります。

22.22.2 Mindlin-Reissner プレートモデルブリックをモデルに追加する

`src/getfem/getfem_linearized_plates.h` で定義されている次の関数は Mindlin-Reissner プレートモデル項を横方向変位 u_3 と回転 $theta$ に追加します

```
size_type add_Mindlin_Reissner_plate_brick
(model, mim, mim_reduced, name_u3, name_theta, param_E,
 param_nu, param_epsilon, param_kappa, variant = 2, region)
```

`name_u3` は横変位を表す変数の名前、`name_theta` は回転を表す変数、`param_E` はヤング係数、`param_nu` は Poisson 比、`param_epsilon` は板厚、`param_kappa` はせん断補正係数です。このブリックは弱形式言語を使用するため、パラメータはこの言語の正規表現にすることができます。3 つの変数があります。`variant = 0` は非還元式に対応し、その場合には積分法 `mim` のみが使用されます。実際にはこの変形は強力なロッキング現象の影響を受けやすいので使用できません。`variant = 1` は回転項に `mim` が使用され、横断せん断項に `mim_reduced` が使用される縮小積分に対応します。`variant = 2`（デフォルト）は横せん断項の回転 RT0 要素への投影に対応します。現時点では、四辺形のみにも適用されています（3 角形要素のロック現象を除去するには不十分であるため）。高次の要素を使用すると、RT0 の投影によって近似の次数が減少することにも注意してください。モデル内の要素のインデックスを返します。

回転 RT0 要素の投影は次の関数によってモデルに追加できます:

```
void add_2D_rotated_RT0_projection(model, transname);
```

22.23 過渡問題の積分のためのモデルツール

時間積分法は各反復で求解する問題を記述することによって model オブジェクトを使用して直接書くことができますが、model オブジェクトはそのような方法の記述を容易にするためのいくつかの基本的なツールを備えています。これらのツールは、次の基本原則に基づいています。

- モデルの元の変数は、現在の時間ステップ（例えば、ステップ n ）で解かれるべきシステムの状態を表します。これは、中点法の場合でさえ、主にシステムの異なる変数に異なる方法を適用する必要がある場合、すべての変数は固有の時間ステップでシステムを記述する必要があるためです。
- いくつかのデータがモデルに追加され、以前の時間ステップでのシステムの状態を表します。古典的な 1 ステップ法（現時点では、1 ステップ法のみが提供される）では、前の時間ステップのみが記憶されます。例えば u が変数である（ステップ n で表される）場合、*Previous_u* は前の時間ステップ（ステップ $n-1$ ）における変数の状態を表すデータです。マルチステップ法への将来的な拡張のために、*Previous2_u* は時間ステップ $n-2$ における変数を表すことができます。
- いくつかの中間変数は、時間導関数（および 2 次問題の 2 次時間導関数）を表すためにモデルに追加されます。例えば、 u が変数であれば、*Dot_u* は u の一次時間導関数を表し、*Dot2_u* は 2 次のもを表します。モデル内のこれらの変数を参照して、要素を追加するか、弱形式言語で使用することができます。しかし、これらは独立変数とはみなされず、時間積分法によって対応する元の変数（affine 方式で）にリンクされます。アルゴリズムの大部分は前の時間ステップで時間微分を必要とし、*Previous_Dot_u* と *Previous_Dot2_u* データをモデルに加えることが可能です。
- 異なる時間積分法は、モデルの各変数に適用することができます。ほとんどの場合、乗数変数、より一般的には時間微分が使用されない変数は時間積分法を必要としないことに注意してください。
- データ t は時間パラメータを表し、（弱形式言語で、またはいくつかの要素のパラメータとして）使用することができます。システムの構築前に、データ t は自動的に時間ステップ n に更新されます。
- 各反復で解くべき問題は、速度と加速度が導入された中間変数によって表されるその自然（弱）定式化における過渡問題の定式化に対応します。例えば、問題

$$\dot{u}(t, x) - \Delta u(t, x) = \sin(t)$$

の弱形式言語による表現は以下のように記述することができます:

```
Dot_u*Test_u + Grad_u.Grad_Test_u - sin(t)*Test_u
```

（もちろん、このような状況では、効率の理由から線形要素の使用が望ましいです）

- 実装されたすべての 1 ステップ法では、時間の問題は 1 次と 2 次の両方の問題（または準静的問題）でも反復から変更することができます。
- 2 次時間問題（1 次時間）に対する計算法は、2 次または 1 次の時間に、あるいは準静的問題（1 次または準

静的問題へ)に適用することもできます。速度/変位に対応する初期データを計算法の回数に関して初期化しなければならないという点を除いて、一次問題の法は、2次問題の時間問題に適用することはできません。

22.23.1 一次問題の陰的 Theta 法

次の問題を見てください

$$M\dot{U} = F(U)$$

ここで、 $F(U)$ は非線形である可能性があります (結合問題の他の変数に依存するかもしれません) dt は時間ステップ、 $V = \dot{U}$ と U^n, V^n は時刻 ndt における U, V の近似です。Theta 法は

$$\begin{cases} U^n = U^{n-1} + dt(\theta V^n + (1-\theta)V^{n-1}), \\ MV^n = F(U^n), \end{cases}$$

Theta 法のパラメータである $\theta \in (0, 1]$ ($\theta = 0$ 法は前方 Euler 法に対応し、陰解法ではありません) そして、 U^{n-1}, V^{n-1} が与えられます。

最初の時間ステップの前に U^0 を初期化する必要があります、しかし、 V^0 も必要です ($\theta = 1$ の場合を除く)。この例では、それは $M^{-1}F(U^0)$ に対応するはずですが、汎用的な結合問題 M は特異であり、汎用的な事前計算 V^0 を得ることは困難です。したがって V^0 もまた準備しなければいけません。代替的に (以下を参照)、`model` オブジェクト (および標準解) は (非常に) 小さな時間ステップで Backward Euler 法を適用することによりそれらを評価するための平均値を計算します。

次の式は時間微分により導き出すことができます:

$$V^n = \frac{U^n - U^{n-1}}{\theta dt} - \frac{1-\theta}{\theta} V^{n-1}$$

この方法をモデルの変数 “u” に適用すると、次の Affine 依存変数がモデルに追加されます:

`"Dot_u"`

これは変数の時間微分を表し、いくつかのブリック定義で使用することができます。

以下のデータも追加されています:

`"Previous_u"`, `"Previous_Dot_u"`

これは前回の時間ステップにおける “u” と “Dot_u” の値に対応します。

解く前にデータ “Previous_u” (例 U^0 を除く) を初期化する必要があります。ここでも、“Previous_Dot_u” は、次のセクションで説明するように、初期化されるか事前計算される必要があります。したがって、“Dot_u” の Affine 依存性は、次のように与えられます。


```
Dot_u = (u - Previous_u)/(theta*dt) - Previous_Dot_u*(1-theta)/theta
```

つまり、“Dot_u”は構築時に“u”（ $1/(\theta * dt)$ を掛けたもの）と前の時間ステップに依存する残りの定数項による表現で置き換えられます。変数へのこの方法の追加は次のように行います。:

```
add_theta_method_for_first_order(model &md, const std::string &varname, scalar_type theta);
```

22.23.2 速度/加速度の事前計算

大部分の時間積分法（例えば、後方 Euler 法を除く）は、最初の時間ステップの前に第 1 または第 2 の時間微分の事前計算を必要とします（例えば、第 1 次問題のための Theta 法、第 2 次問題のための A^0 ...）。

ユーザーはこれらの導関数を初期化するか、モデルに自動的に近似するよう依頼するか選択できます。

補完的な導関数を近似するために（現時点で）使用されている方法は、

$$M\dot{U} = F(U)$$

Theta 法（前のセクションを参照）を使用します。 V_0 を近似するために、Theta 法は、非常に小さな時間ステップで、 $\theta = 1$ （すなわち、後方 Euler 法）に適用されます。これは、後退 Euler が初期時間微分を必要としないので可能です。その後、非常に小さな時間ステップの終了時の後方 Euler のおかげで計算された時間微分は、単純に初期時間微分の近似値として使用されます。

モデル *md* について、以下の命令は

```
model.perform_init_time_derivative(ddt);
standard_solve(model, iter);
```

初期時間導関数の近似を自動的に実行することを可能にします。パラメータ *ddt* は、近似を実行するために使用される小さな時間ステップに対応します。典型的には、 $ddt = dt/20$ を使用します、ここで、*dt* は過渡問題を近似するために使用される時間ステップです（以下の例を参照）。

22.23.3 2 次問題の陰的 Theta 法

次の問題を見てください

$$M\ddot{U} = F(U)$$

dt は時間ステップ、 $V = \dot{U}$, $A = \ddot{U}$ と U^n, V^n, A^n は時刻 ndt における U, V, A のの近似です、Theta 法は次の式になります。

$$\begin{cases} U^n = U^{n-1} + dt(\theta V^n + (1-\theta)V^{n-1}), \\ V^n = V^{n-1} + dt(\theta A^n + (1-\theta)A^{n-1}), \\ MA^n = F(U^n), \end{cases}$$

Theath 法のパラメータは $\theta \in (0, 1]$ であり ($\theta = 0$ 、計算法は前方 Euler 法に対応し、陰解法ではありません) $U^{n-1}, V^{n-1}, A^{n-1}$ が与えられています。

最初のステップでは、 U^0, V^0 が与えられ、 A^0 が与えられるか、あらかじめ計算されている必要があります ($\theta = 1$ を除く)。

次の式は時間微分により導き出すことができます：

$$V^n = \frac{U^n - U^{n-1}}{\theta dt} - \frac{1 - \theta}{\theta} V^{n-1}$$

$$A^n = \frac{U^n - U^{n-1}}{\theta^2 dt^2} - \frac{1}{\theta^2 dt} V^{n-1} - \frac{1 - \theta}{\theta} A^{n-1}$$

この法をモデルの変数 “u” に適用すると、以下の Affine 依存変数がモデルに追加されます：

`"Dot_u", "Dot2_u"`

変数の 1 次および 2 次の時間微分を表し、いくつかのブリック定義で使用できます。

以下のデータも追加されています：

`"Previous_u", "Previous_Dot_u", "Previous_Dot2_u"`

前回の時間ステップにおける “u”、“Dot_u”、“Dot2_u” の値に対応します。

解く前に、データ “Previous_u” と “Previous_Dot_u” (例では U^0) を初期化し、“Previous_Dot2_u” を初期化したは事前計算する必要があります ($\theta = 1$)。したがって、Affine 依存性は次のように与えられます。

`Dot_u = (u - Previous_u) / (theta*dt) - Previous_Dot_u*(1-theta)/theta`

`Dot2_u = (u - Previous_u) / (theta*theta*dt*dt) - Previous_Dot_u / (theta*theta*dt) - Previous_Dot2_u*(1-`

変数へのこの計算法の追加は次のように行います。

```
add_theta_method_for_second_order(model &md, const std::string &varname,
                                   scalar_type theta);
```

22.23.4 2 次問題の陰的 Newmark 法

次の問題を見てください

$$M\ddot{U} = F(U)$$

dt は時間ステップ、 $V = \dot{U}$, $A = \ddot{U}$ と U^n, V^n, A^n は時刻 ndt における U, V, A のの近似です、Theta 法は次の式になります。

$$\begin{cases} U^n = U^{n-1} + dtV^n + \frac{dt^2}{2}(2\beta V^n + (1 - 2\beta)V^{n-1}), \\ V^n = V^{n-1} + dt(\gamma A^n + (1 - \gamma)A^{n-1}), \\ MA^n = F(U^n), \end{cases}$$

$\beta \in (0, 1]$ と $\gamma \in [1/2, 1]$ は Newmark 法のパラメータであり、 $U^{n-1}, V^{n-1}, A^{n-1}$ が与えられています。

最初のステップでは、 U^0, V^0 が与えられていなければいけません。そして A^0 が与えられているか事前計算されている必要があります ($\beta = 1/2, \gamma = 1$ を除きます)。

次の式は時間微分により導き出すことができます：

$$V^n = \frac{\gamma}{\beta dt}(U^n - U^{n-1}) + \frac{\beta - \gamma}{\beta} V^{n-1} + dt(1 - \frac{\gamma}{2\beta}) A^{n-1}$$

$$A^n = \frac{U^n - U^{n-1}}{\beta dt^2} - \frac{1}{\beta dt} V^{n-1} - (1/2 - \beta) A^{n-1}$$

この法をモデルの変数 “u” に適用すると、以下の Affine 依存変数がモデルに追加されます：

"Dot_u", "Dot2_u"

変数の 1 次および 2 次の時間微分を表し、いくつかのブリック定義で使用できます。

以下のデータも追加されています：

"Previous_u", "Previous_Dot_u", "Previous_Dot2_u"

前回の時間ステップにおける “u”、“Dot_u”、“Dot2_u” の値に対応します。

最初の球解の前に、データ “Previous_u” と “Previous_Dot_u” (この例では U^0 に対応) を初期化する必要があります。データ “Previous_Dot2_u” は与えられているか、事前計算されています (速度/加速度の事前計算を参照してください $\beta = 1/2, \gamma = 1$ を除きます)。

変数へのこの計算法の追加は次のように行います。

```
add_Newmark_scheme(model &md, const std::string &varname,
                    scalar_type beta, scalar_type gamma);
```

22.23.5 一時的な項

これまでのセクションで説明したように、法が適用される変数の時間微分を表すために、いくつかの中間変数がモデルに追加されています。再度、「u」がそのような変数である場合、「Dot_u」は、使用された法によって近似された「u」の時間微分を表す。

これはまた、一時的な項を表現するために “Dot_u” (および時間的に 2 つの問題の順序で “Dot2_u”) を使用できることを意味します。弱形式言語では、

$$\int_{\Omega} \dot{u} v dx$$

次のように単純に表現することができます。

```
Dot_u*Test_u
```

同様に、*GetFEM++* は “Dot_u” に適用できます。たとえば、次のような場合です。

```
getfem::add_mass_brick(model, mim, "Dot_u");
```

同じ過渡項を追加します。

非常に重要： “Dot_u” のような Affine 依存変数に適用された既存のモデルブリックを追加する場合、対応する試行関数が、対応する元の変数（ここでは “Test_u”）のものであると常に仮定されます。言い換えれば、速度に対応する試行変数である “Test_Dot_u” は使用されません。これは、試行関数が元の変数に対応するように、元の変数の項で問題を解決するために行われた選択に対応します。

Kelvin-Voigt 線形化粘度項を説明するために使用できるモデルブリックの別の例は、線形弾性ブリックとなります。

```
getfem::add_isotropic_linearized_elasticity_brick(model, mim, "Dot_u", "lambda_viscosity", "mu_viscosity");
```

2 次の過渡の弾力性の問題に適用するときに適用します。

22.23.6 時間ステップのシーケンスに対する計算

一般に、異なる時間ステップでの解法は次の形式をとります:

```
for (scalar_type t = 0.; t < T; t += dt) { // time loop

    // Eventually compute here some time dependent terms

    iter.init();
    getfem::standard_solve(model, iter);

    // + Do something with the solution (plot or store it)

    model.shift_variables_for_time_integration();
}
```

メソッドの呼び出し:

```
model.shift_variables_for_time_integration();
```

変数 *u* と *Dot_u* の現在の値を前のもの (*Previous_u* と *Previous_Dot_u*) にコピーするので、2 つの時間ステップの間に必要です。

22.23.7 境界条件

もちろん、標準的な境界条件は、未知の様々な変数に通常適用することができます。デフォルトでは、Dirichlet、Neumann または接触境界条件を未知数に適用するということは、現在の時間ステップ n で変数に条件が規定されていることを意味します。

22.23.8 小さな例：熱方程式

GetFEM++ ディストリビューションの完全なコンパイル可能なプログラム tests/heat_equation.cc がテストプログラムに対応しています。matlab インタフェースの 2 次時間的ステップ問題の例については、/interface/tests/matlab/demo_wave_equation.m を参照してください。

mf_u と mim が有効なメッシュに定義された有効な有限要素と積分方法であると仮定すると、次のコードは単位拡散係数を仮定したメッシュ上の温度の進展を近似します。

```
getfem::model model;
model.add_fem_variable("u", mf_u, 2); // Main unknown of the problem

getfem::add_generic_elliptic_brick(model, mim, "u"); // Laplace term

// Volumic source term.
getfem::add_source_term_generic_assembly_brick(model, mim, "sin(t)*Test_u");

// Dirichlet condition.
getfem::add_Dirichlet_condition_with_multipliers
    (model, mim, "u", mf_u, DIRICHLET_BOUNDARY_NUM);

// transient part.
getfem::add_theta_method_for_first_order(model, "u", theta);
getfem::add_mass_brick(model, mim, "Dot_u");

gmm::iteration iter(residual, 0, 40000);

model.set_time(0.); // Init time is 0 (not mandatory)
model.set_time_step(dt); // Init of the time step.

// Null initial value for the temperature.
gmm::clear(model.set_real_variable("Previous_u"));

// Automatic computatio of Previous_Dot_u
model.perform_init_time_derivative(dt/20.);
iter.init();
standard_solve(model, iter);
```

```
// Iterations in time
for (scalar_type t = 0.; t < T; t += dt) {

    iter.init();
    getfem::standard_solve(model, iter);

    // + Do something with the solution (plot or store it)

    // Copy the current variables "u" and "Dot_u" into "Previous_u"
    // and "Previous_Dot_u".
    model.shift_variables_for_time_integration();
}
```

22.23.9 陰的/陽的な項

...

22.23.10 陽解法

...

22.23.11 時間ステップ適応

...

22.23.12 準静的問題

...

22.24 摩擦ブリックとの微小すべり接触

これらのブリックの目的は、剛性のある基礎または2つの弾性構造の間の弾性構造の摩擦を伴うかまたは伴わない接触条件を考慮に入れることです。これらのブリックの接触は小さな変形近似に制限されています（これは平坦な障害物に大きな変形を含むことがあります）。

22.24.1 接触の近似

小さな変形問題に対しては、接線方向の変位が通常の変位に影響を与えない場合、摩擦条件との接触の単純な表現（有限変形と比較して！）が通常使用されます。これは、障害物が完全に平坦でない場合には、当然、接線方向の変位が接点が保持する点に影響するという意味での近似です。これは、接触状態が基準構成に記載されているとみなすことができる小さな変形の場合には当てはまりません。

主に摩擦条件との接触の離散化は、結合接触条件（通常は変位有限要素節点で規定される）または微小すべり接触条件（通常は乗数有限要素節点で規定される）の2つです。2つの離散化は同様のシステムにつながります。しかし、量の解釈は同じではありません。第3のアプローチは、弱積分接触条件である **Getfem** 接触ブリックです。各反復時に接触境界に非線形積分の計算が必要ですが、数値分解能は連続原理から直接導出されるため、潜在的によりスケラブルです。

詳細は、例えば [KI-OD1988]、[KH-PO-RE2006] および [LA-RE2006] で見つけることができます。

22.24.2 結合接触状態

節点接触条件は、摩擦条件との接触（または摩擦条件なし）が適用されるいくつかの接触節点 $a_i, i = 1..N_c$ で構成されます。接触条件は次の通りです。

$$u_N(a_i) - \text{gap}_i \leq 0, \quad \lambda_N^i \leq 0, \quad (u_N(a_i) - \text{gap}_i)\lambda_N^i = 0,$$

ここで、 λ_N^i は、 a_i の等価節点接触力です。おして、 $u_N(a_i)$ は弾性体と障害物の間、または2つの弾性体の正規相対変位です。項 gap_i は、参照設定の2つのソリッド間の通常のギャップを表します。摩擦条件は

$$\begin{aligned} \|\lambda_T^i\| &\leq -\mathcal{F}\lambda_N^i, \\ \lambda_T^i &= \mathcal{F}\lambda_N^i \frac{\dot{u}_T}{\|\dot{u}_T\|} \quad \text{when } \dot{u}_T \neq 0, \end{aligned}$$

ここで、 \dot{u}_T は相対すべり速度です。 \mathcal{F} は摩擦係数で、 λ_T^i は a_i 上の等価節点摩擦力です。摩擦条件は、包含物によって整理することができます。

$$\lambda_T^i \in \mathcal{F}\lambda_N^i \text{Dir}(\dot{u}_T),$$

ここで、 $\text{Dir}(\dot{u}_T)$ は $x \mapsto \|x_T\|$ (すなわち、 $x \neq 0$ で $\text{Dir}(0)$ である閉じた単位円) の劣微分の多値関数となります。2次元の場合、 $\text{Dir}(\dot{u}_T)$ は以下のようにになります $\text{Sign}(\dot{u}_T)$ 。ここで Sign は多値記号写像です。

摩擦が発生する接触の完全な線形弾性問題は、

増大パラメータ r が与えられた場合、接触および摩擦条件は、等価的に投影の項で表すことができます。

$$\begin{aligned} \frac{1}{r}(\lambda_N^i - P_{]-\infty, 0]}(\lambda_N^i - r(u_N(a_i) - \text{gap}_i))) &= 0, \\ \frac{1}{r}(\lambda_T^i - P_{\mathcal{B}(-\mathcal{F}P_{]-\infty, 0]}(\lambda_N^i - r(u_N(a_i) - \text{gap}_i))})(\lambda_T^i - r\dot{u}_T(a_i))) &= 0, \end{aligned}$$

P_K は K 凸包の投影で、 $\mathcal{B}(-\mathcal{F}\lambda_N^i)$ は中心 0 半径 $-\mathcal{F}\lambda_N^i$ の球です。これらの表現は、半 Smooth Newton 法を実行するために使用されます。

摩擦との接触に提出された線形弾性問題を近似したとします。そして、もし U が変位のための未知数のベクトルなら、次のような行列 B_N と B_T を表現することができます。

$$u_N(a_i) = (B_N U)_i,$$

$$(\dot{u}_T(a_i))_k = (B_T \dot{U})_{(d-1)(i-1)+k},$$

ここで、 d は領域の次元であり、 $k = 1..d-1$ です。摩擦との接触による弾性問題の表現は、次のように書くことができます。

$$KU = L + B_N^T \lambda_N + B_T^T \lambda_T,$$

$$-\frac{1}{r\alpha_i} (\lambda_N^i - P_{]-\infty, 0]}(\lambda_N^i - \alpha_i r ((B_N U)_i - \text{gap}_i))) = 0, \quad i = 1..N_c,$$

$$-\frac{1}{r\alpha_i} (\lambda_T^i - P_{\mathcal{B}(-\mathcal{F}P_{]-\infty, 0]}(\lambda_N^i - \alpha_i r ((B_N U)_i - \text{gap}_i)))}(\lambda_T^i - \alpha_i r (B_T U - B_T U^0)_i)) = 0, \quad i = 1..N_c,$$

ここで、 α_i は引数の均質化のために追加できるパラメータです。 $(B_T U)_i$ はインデックス $(d-1)(i-1)+1$ から $(d-1)i$ へのサブベクトルであり、シンプルさのために、滑り速度 $B_T \dot{U}$ が前の時間ステップの変位 U^0 によって、 $\frac{(B_T U - B_T U^0)}{\Delta t}$ に離散化されています。もちろん他の滑り速度の表現も可能であるが、時間ステップ Δt は摩擦条件の中に表れない。なぜなら、それは滑り速度の方向に影響を与えないためです。

その場合、homogenization 係数 α_i は、式 h^{d-2} に比例して取ることができます (h はブリックの直径です。)。このようにして、増強パラメータ r は、 N/m^2 で表すことができ、弾性体の Young 率に近づくように選択されています。この解は、増強パラメータの値に対して敏感ではないことに注意してください。

22.24.3 微小すべり接触条件

結合接触条件にはいくつかの欠点があります：ロック現象、過度の制約。実際には、変位と比較して減少した次数の乗数を使用することは、しばしばより安定しており、同じ精度となります（結合接触条件は、変位と同じ有限要素法で記述された乗数にほぼ相当します）。

φ_i を、変位を記述する有限要素の試行関数とします。 ψ_i は、接触境界 Γ_c 上の乗数を記述する有限要素の試行関数です。法線応力を記述する許容乗数の集合は、

$$\Lambda_N^h = \{\mu_N^h = \sum \mu_N^j \psi_j : \mu_N^h(a_i) \leq 0, \quad i = 1..N_c\}$$

ここで、 a_i 、 $i = 1..N_c$ は、乗数に対応する有限要素節点です。離散接触条件は、次のように弱形で表されます。

$$\int_{\Gamma_c} (\mu_N^h - \lambda_N^h)(u_N - \text{gap}) d\Gamma \geq 0 \quad \forall \mu_N^h \in \Lambda_N^h.$$

その場合、成分 λ_N^i は接触応力 (N/m^2) であり、行列 B_N は次のように表すことができます。

$$(B_N)_{ij} = \int_{\Gamma_c} \psi_i \varphi_j d\Gamma.$$

行列 B_T も同様の方法で書くことができます。摩擦状態は弱形で書くことができます。

$$\int_{\Gamma_c} (\mu_T^h - \lambda_T^h) \dot{u}_T d\Gamma \geq 0 \quad \forall \mu_T^h \in \Lambda_T^h(\mathcal{F} \lambda_N^h),$$

ここで、 $\Lambda_T^h(\mathcal{F} \lambda_N^h)$ は許容できる摩擦応力の離散集合です。

最後に、直接結節接触状態の表現を再掲します

$$\begin{aligned} KU &= L + B_N^T \lambda_N + B_T^T \lambda_T, \\ -\frac{1}{r\alpha_i} (\lambda_N^i - P_{[-\infty, 0]}(\lambda_N^i - \alpha_i r((B_N U)_i - \text{gap}_i))) &= 0, \quad i = 1..N_c, \\ -\frac{1}{r\alpha_i} (\lambda_T^i - P_{\mathcal{B}(-\mathcal{F} P_{[-\infty, 0]}(\lambda_N^i - \alpha_i r((B_N U)_i - \text{gap}_i)))}(\lambda_T^i - \alpha_i r(B_T U - B_T U^0)_i)) &= 0, \quad i = 1..N_c, \end{aligned}$$

それを除いて、 λ_N^i と λ_T^i は力の密度であり、そして α_i は $1/h^d$ に比例するように選択され、増強パラメータ r は弾性体の Young 率に近いまま選択できるようにします。

追加の安定化技術（[HI-RE2010] を参照）がないと、変位の有限要素と乗数の要素の間で極限条件を満たされなければならないことに注意してください。これは、乗数の有限要素が、変位の有限要素よりも “less rich” でなければならないことを特に意味します。

22.24.4 弱積分接触条件

弱積分接触式は、離散的な許容応力の集合を明示的に記述しません。摩擦条件との接触のための汎用的な *Nitsche* の方法も参照してください。接触応力（摩擦を含む）は接触境界 Γ_c 上の有限要素空間 W^h 上に記述されています。

$$\lambda^h \in W^h = \left\{ \sum \lambda_i \psi_i, \lambda_i \in \mathbb{R}^d \right\}$$

d は問題の次元であり、 ψ_i は接触応力が発生する形状関数です。今、外向きの単位ベクトル n を与えられます接触境界 Γ_c 上の（通常は障害物の法線）の標準分解を行います。

$$\lambda_N^h = \lambda^h \cdot n, \quad \lambda_T^h = \lambda^h - \lambda_N^h n, \quad u_N^h = u^h \cdot n, \quad u_T^h = u^h - u_N^h n,$$

ここで、 u^h は、有限要素空間で近似された変位場 V^h です。これにより、接触状態を以下のように表現することができます。

$$\int_{\Gamma_c} (\lambda_N^h + (\lambda_N^h - r(u_N^h - \text{gap}))_-) \mu_N^h d\Gamma = 0 \quad \forall \mu^h \in W^h,$$

ここで、 gap は、参照構成で与えられた最初のギャップです。 r は拡大パラメータで、 $(\cdot)_- : \mathbb{R} \rightarrow \mathbb{R}_+$ は負の部分です。摩擦条件も同様に書くことができます。

$$\int_{\Gamma_c} (\lambda_T^h - P_{\mathcal{B}(\lambda_N^h - r(u_N^h - \text{gap}))_-}(\lambda_T^h - r\alpha(u_T^h - w_T^h))) \cdot \mu_T^h d\Gamma = 0 \quad \forall \mu^h \in W^h,$$

ここで、 $B(\rho)$ は、中心 0 と半径 ρ と投影 $P_{B(\rho)}$ の閉じた円です。(慣性によって、ボールは元の位置 $\rho \leq 0$ に戻ります)。以下の項 $\alpha(u_T^h - w_T^h)$ は、摺動速度の近似値を表しています。パラメータ α とフィールド w_T^h は、選択された近似に関して適合させなければなりません。例えば、標準的な有限差分は次のようになります。

$$(\dot{u}_T^h)^{n+1} \approx \frac{(u_T^h)^{n+1} - (u_T^h)^n}{dt}$$

$\alpha = 1/dt$ と $w_T^h = (u_T^h)^n$ を選択する必要があります。ボールの対称性のために、パラメータ α は、調合において重要な役割を果たさないことに注意してください。これは、接触条件のための増強パラメータと摩擦条件のための増強パラメータとの間のスケールリングとして単純に見ることができます。以前の接触式とは対照的に、ここでは、増強パラメータ（独立性は連続レベルでのみ生じる）に関して条件の厳密な独立性はありません。

GetFEM++ ブリックは、Alart-Curnier 拡張 Lagrangian 配合 [AL-CU1991] から得られた接触条件の 4 つのバージョンを実装します。最初のものは非対称バージョンに対応します。それは次のようになります。

$$\left\{ \begin{array}{l} a(u^h, v^h) + \int_{\Gamma_c} \lambda^h \cdot v^h d\Gamma = \ell(v^h) \quad \forall v^h \in V^h, \\ -\frac{1}{r} \int_{\Gamma_c} (\lambda_N^h + (\lambda_N^h - r(u_N^h - gap))_-) \mu_N^h d\Gamma \\ \quad - \frac{1}{r} \int_{\Gamma_c} (\lambda_T^h - P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h))) \cdot \mu_T^h d\Gamma = 0 \quad \forall \mu^h \in W^h, \end{array} \right.$$

ここで、 $a(\cdot, \cdot)$ と $\ell(v)$ は問題の残りの部分を表します。 u (例えば線形弾性) $\rho = \mathcal{F}(\lambda_N^h - r(u_N^h - gap))_-$ 。この場合、数学的分析は、値 $r = r_0/r$ により増強パラメータの値 $r = r_0/r$ を選択することにあたることに注意してください。 r_0 は、弾性係数の次元を有します (古典的には Young 率の値)。Newton 反復を書くためには、接線系を導出しなければなりません。それは右側ではなく、接触と摩擦の項のみで書くことができます。

$$\left\{ \begin{array}{l} \dots - \int_{\Gamma_c} \delta_\lambda \cdot v d\Gamma = \dots \quad \forall v^h \in V^h, \\ -\frac{1}{r} \int_{\Gamma_c} (1 - H(r(u_N^h - gap) - \lambda_N)) \delta_{\lambda_N} \mu_N^h d\Gamma - \int_{\Gamma_c} H(r(u_N^h - gap) - \lambda_N) \delta_{u_N} \mu_N^h d\Gamma \\ \quad - \frac{1}{r} \int_{\Gamma_c} (\delta_{\lambda_T} - D_x P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h))) \delta_{\lambda_T} \cdot \mu_T^h d\Gamma \\ \quad - \int_{\Gamma_c} \alpha D_x P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{u_T} \cdot \mu_T^h d\Gamma \\ \quad + \int_{\Gamma_c} (\mathcal{F} D_\rho P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{u_N}) \cdot \mu_T^h d\Gamma \\ \quad - \int_{\Gamma_c} (\frac{\mathcal{F}}{r} D_\rho P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{\lambda_N}) \cdot \mu_T^h d\Gamma = \dots \quad \forall \mu^h \in W^h, \end{array} \right.$$

ここで、 $H(\cdot)$ は Heaviside 関数です (負の引数は 0、非負の引数は 1)、 $D_x P_{B(\rho)}(x)$ と $D_\rho P_{B(\rho)}(x)$ は、($\rho \leq 0$ が消滅すると想定される) 射影 $B(\rho)$ の微分と δ_u は、接線問題に対応する未知数です。

第 2 のバージョンは、“対称”バージョンに対応します。実際には、摩擦のない場合のみ対称です (この場合、こ

これは、拡張された Lagrangian 式から直接導かれるからである)。それは以下の通りになります。

$$\left\{ \begin{array}{l} a(u^h, v^h) + \int_{\Gamma_c} (\lambda_N^h - r(u_N^h - gap))_ - v_N^h d\Gamma \\ \quad - \int_{\Gamma_c} P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \cdot v_T^h d\Gamma = \ell(v^h) \quad \forall v^h \in V^h, \\ -\frac{1}{r} \int_{\Gamma_c} (\lambda_N^h + (\lambda_N^h - r(u_N^h - gap))_ -) \mu_N^h d\Gamma \\ \quad - \frac{1}{r} \int_{\Gamma_c} (\lambda_T^h - P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h))) \cdot \mu_T^h d\Gamma = 0 \quad \forall \mu^h \in W^h, \end{array} \right.$$

接線系

$$\left\{ \begin{array}{l} \dots + \int_{\Gamma_c} rH(r(u_N^h - gap) - \lambda_N) \delta_{u_N} v_N - H(r(u_N^h - gap) - \lambda_N) \delta_{\lambda_N} v_N d\Gamma \\ \quad + \int_{\Gamma_c} r\alpha D_x P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{u_T} \cdot v_T^h d\Gamma \\ \quad - \int_{\Gamma_c} D_x P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{\lambda_T} \cdot v_T^h d\Gamma \\ \quad - \int_{\Gamma_c} (r\mathcal{F} D_\rho P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{u_N}) \cdot v_T^h d\Gamma \\ \quad - \int_{\Gamma_c} (\mathcal{F} D_\rho P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{\lambda_N}) \cdot v_T^h d\Gamma = \dots \quad \forall v^h \in V^h, \\ -\frac{1}{r} \int_{\Gamma_c} (1 - H(r(u_N^h - gap) - \lambda_N)) \delta_{\lambda_N} \mu_N^h d\Gamma - \int_{\Gamma_c} H(r(u_N^h - gap) - \lambda_N) \delta_{u_N} \mu_N^h d\Gamma \\ \quad - \frac{1}{r} \int_{\Gamma_c} (\delta_{\lambda_T} - D_x P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{\lambda_T}) \cdot \mu_T^h d\Gamma \\ \quad - \int_{\Gamma_c} \alpha D_x P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{u_T} \cdot \mu_T^h d\Gamma \\ \quad + \int_{\Gamma_c} (\mathcal{F} D_\rho P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{u_N}) \cdot \mu_T^h d\Gamma \\ \quad - \int_{\Gamma_c} (\frac{\mathcal{F}}{r} D_\rho P_{B(\rho)}(\lambda_T^h - r\alpha(u_T^h - w_T^h)) \delta_{\lambda_N}) \cdot \mu_T^h d\Gamma = \dots \quad \forall \mu^h \in W^h, \end{array} \right.$$

$\rho = \mathcal{F}(\lambda_N^h - r(u_N^h - gap))_ -$ のようになります。

第 3 のバージョンは、ペナルティ化された接触および摩擦状態に対応します。乗算器を使用する必要はありません。このバージョンでは、パラメータ r はペナルティ化パラメータであり、非貫通および Coulomb 摩擦条件の良好な近似を実行するのに十分な大きさです。処置は以下の通りです。

$$\left\{ \begin{array}{l} a(u^h, v^h) + \int_{\Gamma_c} r(u_N^h - gap)_ + v_N^h d\Gamma \\ \quad + \int_{\Gamma_c} P_{B(\mathcal{F}r(u_N^h - gap)_ +)}(r\alpha(u_T^h - w_T^h)) \cdot v_T^h d\Gamma = \ell(v^h) \quad \forall v^h \in V^h, \end{array} \right.$$

接線系

$$\left\{ \begin{array}{l} \dots + \int_{\Gamma_c} rH(u_N^h - gap) \delta_{u_N} v_N d\Gamma \\ \quad - \int_{\Gamma_c} r\alpha D_x P_{B(\mathcal{F}r(u_N^h - gap)_ +)}(r\alpha(u_T^h - w_T^h)) \delta_{u_T} \cdot v_T^h d\Gamma \\ \quad + \int_{\Gamma_c} (r\mathcal{F}H(u_N^h - gap) D_\rho P_{B(\mathcal{F}r(u_N^h - gap)_ +)}(r\alpha(u_T^h - w_T^h)) \delta_{u_N}) \cdot v_T^h d\Gamma = \dots \quad \forall v^h \in V^h, \end{array} \right.$$

22.24.5 数値継続

また、*GetFEM++* は積分された弱接触条件に基づいて、離散化された進化的接触問題の数値解を見つけるための数値継続法を開発しています（一般的な紹介については、[数値連続法と分岐](#) を参照）。この目的のために、パラメータに依存する摺動速度を摩擦条件に加えることができます。

$$\int_{\Gamma_c} \left(\lambda_T^h - P_{B(-\mathcal{F}\lambda_N^h)}(\lambda_T^h - r(\alpha(u_T^h - w_T^h) + (1 - \gamma)z_T^h)) \right) \cdot \mu_T^h d\Gamma = 0 \quad \forall \mu^h \in W^h.$$

ここで、 γ はパラメータであり、 z_T^h は初期滑り速度です。もし誰かが選んだのであれば、

$$\alpha = \frac{1}{dt}, \quad w_T^h = (u_T^h)^n, \quad z_T^h = \frac{(u_T^h)^n - (u_T^h)^{n-1}}{dt},$$

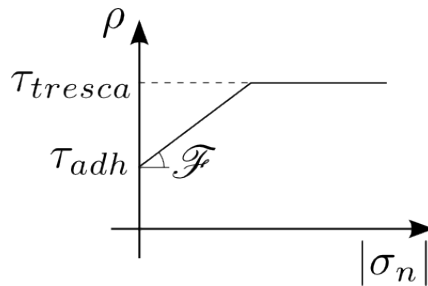
t_n と t_{n+1} の標準摩擦条件をそれぞれ 0 と 1 に戻します。

22.24.6 摩擦則

純粋な Coulomb 摩擦とは別に、次のようになります。 $\rho = \mathcal{F} |\sigma_n|$ 、弱積分接触先フレームワーク *GetFEM++* より汎用的な摩擦法の記述もサポートしています。

$$\rho = \begin{cases} \tau_{adh} + \mathcal{F} |\sigma_n| & \text{if } \tau_{adh} + \mathcal{F} |\sigma_n| < \tau_{tresca} \\ \tau_{tresca} & \text{otherwise} \end{cases}$$

\mathcal{F} は摩擦係数であり、次のようになります。 τ_{adh} は接着荷重（負荷に依存しない）せん断応力であり、次のようになります。 τ_{tresca} は最大せん断応力限界です。



22.24.7 モデルに摩擦を伴うか伴わない接触を加えます

22.24.8 摩擦のない基本接触ブリック

摩擦のない接触ブリックを追加するには、`model` オブジェクトメソッドを呼び出します。

```
getfem::add_basic_contact_brick
    (md, varname_u, multname_n, dataname_r, BN, dataname_gap, dataname_alpha, aug_version);
```

この関数は、乗数変数 $multname_n$ により、 $varname_u$ に摩擦のない接触ブリックを追加します。次のような場合 U は片方向拘束が適用される自由度のベクトルです。行列 B_N は、この条件が $B_N U \leq 0$ のように定義されていなければなりません。制約は乗数 $multname_n$ に与えられています。その次元 B_N は行の数と同じでなければなりません。変数 $dataname_r$ は拡張パラメータの名前です r は許容可能な値の範囲で選択されるべきです。 $dataname_gap$ は初期ギャップを表すオプションのパラメータです。これは、単一の値または値のベクトルにすることができます。 $dataname_alpha$ は、拡張パラメータのためのオプションの均質化パラメータです。

パラメータ $aug_version$ は、非対称の Alart-Curnier 拡張 Lagrange の場合は 1、対称の場合は 2、拡張された乗数に基づく非対称の方法の場合は 3 です。

基本的な接触行列を変更することは可能です B_N を使用して次の通りです。

```
getfem::contact_brick_set_BN(md, indbrick);
```

22.24.9 摩擦を伴う基本接触ブリック

```
getfem::add_basic_contact_brick (md, varname_u, multname_n, multname_t, dataname_r, BN,
    dataname_friction_coeff, dataname_gap, dataname_alpha, aug_version);
```

この関数は 2 つの乗数変数 $multname_n$ と $multname_t$ のにより、 $varname_u$ に摩擦を伴う接触先ブリックを追加します。 U が条件が適用される自由度のベクトルである場合、 B_N の行列は、接触条件が以下のように定義されていなければなりません。 $B_N U \leq gap$ と B_T は、相対接線変位が $B_T U$ となるようにしなければなりません。行列 B_T には、 B_N に $d-1$ を掛けたものがたくさんあります。ここで、 d は領域次元です。接触条件は、次元が B_N の行数に等しく、摩擦条件が乗数 $multname_t$ によって行なわれる乗数 $multname_n$ に与えられています。サイズは行数でなければなりません B_T のパラメータ $dataname_friction_coeff$ は摩擦係数を記述します。それは、スカラまたは各接触条件の係数を記述するベクトルであってもよいです。増補パラメータ r は許容可能な値の範囲で選択する必要があります (Getfem のユーザーマニュアルを参照)。 $dataname_gap$ は初期ギャップを表すオプションのパラメータです。これは、単一の値または値のベクトルにすることができます。 $dataname_alpha$ は、拡張パラメータのためのオプションの均質化パラメータです。

パラメータ $aug_version$ は、非対称の Alart-Curnier 拡張 Lagrangian の場合は 1、対称の場合は 2、拡張乗算器に基づく非対称法の場合は 3、De Saxce の拡張乗算器に基づく非対称投影の場合は 4 です。

基本的な接触行列 B_N と B_T を次のように変更することができます。

```
getfem::contact_brick_set_BN(md, indbrick);
getfem::contact_brick_set_BT(md, indbrick);
```

22.24.10 頑丈な障害物ブリックとの摩擦のない節点接触

```
getfem::add_nodal_contact_with_rigid_obstacle_brick (md, mim, varname_u, multname_n, dataname_r,
  region, obstacle, aug_version);
```

この関数は、モデルに堅い障害物を伴う結合接触摩擦無し接触条件を追加します。この条件は、`region` に対応する境界上の変数 `varname_u` に適用されます。剛体の障害物は、障害物までの符号付き距離である `obstacle` の文字列で記述する必要があります。この文字列は、座標が 'x'、'y' の 2D、'x'、'y'、'z' の 3D 表現でなければなりません。例えば、剛体の障害物が $z \leq 0$ に対応する場合、対応する符号付き距離は単に 'z' になります。`multname_n` は、サイズが境界 `region` の自由度の数である固定サイズの変数でなければなりません。これは接触等価節点力を表します。増強パラメータ `r` は許容可能な値の範囲で選択する必要があります（弾性体の Young 率に近いです、Getfem のユーザーマニュアルを参照）。非対称の Alart-Curnier 拡張 Lagrange の場合は 1、対称の場合は 2、拡張された乗算器に基づく非対称の方法の場合は 3 です。

22.24.11 摩擦を伴う頑丈な障害物のブリックとの節の接触

```
getfem::add_nodal_contact_with_rigid_obstacle_brick (md, mim, varname_u, multname_n, multname_t,
  dataname_r, dataname_friction_coeff, region, obstacle, aug_version);
```

この関数は、モデルに堅い障害物を伴う摩擦条件付きの結合接触を追加します。この条件は、`region` に対応する境界上の変数 `varname_u` に適用されます。剛体の障害物は、障害物までの符号付き距離である `obstacle` 文字列で記述する必要があります。この文字列は、座標が 'x'、'y' の 2D、'x'、'y'、'z' の 3D 表現でなければなりません。例えば、剛体の障害物が $z \leq 0$ に対応する場合、対応する符号付き距離は単に 'z' になります。`multname_n` は、サイズが境界 `region` の自由度の数である固定サイズの変数でなければなりません。接触等価節力を表す。`multname_t` は、サイズが境界 `region` の自由度 $d-1$ の倍数である固定サイズの変数でなければなりませんここで、 d は領域次元です。これは摩擦等価節力を表します。増強パラメータ `r` は許容可能な値の範囲で選択する必要があります（弾性体の Young 率に近いです、Getfem のユーザーマニュアルを参照）。`dataname_friction_coeff` は摩擦係数です。それは、各接点節点上の摩擦係数を表す値のスカラまたはベクトルです。

パラメータ `aug_version` は、非対称の Alart-Curnier 拡張 Lagrangian の場合は 1、対称の場合は 2、拡張乗算器に基づく非対称法の場合は 3、De Saxce の拡張乗算器に基づく非対称投影の場合は 4 です。

22.24.12 非整合メッシュブリック間の摩擦のない節点接触

```
getfem::add_nodal_contact_between_nonmatching_meshes_brick (md, mim1, mim2, varname_u1, varname_u2,
  multname_n, dataname_r, rg1, rg2, slave1=true, slave2=false, aug_version=1);
```

この関数は、1 つまたは 2 つの弾性体の 2 つの面の間に摩擦のない接触状態を追加します。条件は、変数

`varname_u` または変数 `varname_u1` と `varname_u2` に 1 つまたは 2 つの異なる変位フィールドが与えられているかどうかに応じて適用されます。ベクトル `rg1` および `rg2` は、互いに接触することが予想される領域の対を含みます。一面あたり単一の領域の場合、`rg1` と `rg2` は通常の整数として与えられます。単一変位変数の場合、`rg1` と `rg2` の両方で定義された領域は変数 `varname_u` を参照します。2 つの変位変数の場合、`rg1` は `varname_u1` を、`rg2` は `varname_u2` を参照します。`multname_n` は、サイズが `rg1` と `rg2` で定義されたもののうち、スレーブとして特徴づけられるもののうち、それらの領域の自由度の数である固定サイズの変数でなければなりません。接触等価節力を表す、増大パラメータ `r` は、受容可能な値の範囲で選択されるべきです（弾性体の Young 率に近く、Getfem のユーザ文書を参照）。オプションパラメータ `slave1` と `slave2` は、`rg1` と `rg2` で定義された領域が対応して“スレーブ”とみなされた場合に宣言します。デフォルトで `slave1` は真で、`slave2` は false です。すなわち `rg1` はスレーブ面を含み、`rg2` はマスター面です。`slave1` と `slave2` のうちの 1 つだけが true に設定されるのが望ましいです。

パラメータ `aug_version` は、非対称の Alart-Curnier 拡張 Lagrange の場合は 1、対称な Lagrangian の場合は 1、拡張された乗数を使用する非対称の方法の場合は 3 です。

基本的には、このブリックは行列 B_N とベクトルのギャップとアルファを計算し、基本的な接触ブリックを呼び出します。

22.24.13 非整合メッシュと摩擦との間の節点接触

```
getfem::add_nodal_contact_between_nonmatching_meshes_brick
```

```
(md, mim1, mim2, varname_u1, varname_u2, multname_n, multname_t, dataname_r,
  dataname_friction_coeff, rg1, rg2, slave1=true, slave2=false, aug_version=1);
```

この関数は、1 つまたは 2 つの弾性体の 2 つの面の間に摩擦状態の接触を追加します。条件は、変数 `varname_u` または変数 `varname_u1` と `varname_u2` に 1 つまたは 2 つの異なる変位フィールドが与えられているかどうかに応じて適用されます。ベクトル `rg1` および `rg2` は、互いに接触することが予想される領域の対を含みます。一面あたり単一の領域の場合、`rg1` と `rg2` は通常の整数として与えられます。単一変位変数の場合、`rg1` と `rg2` の両方で定義された領域は変数 `varname_u` を参照します。2 つの変位変数の場合、`rg1` は `varname_u1` を、`rg2` は `varname_u2` を参照します。`multname_n` は、サイズが `rg1` と `rg2` で定義されたもののうち、スレーブとして特徴づけられるもののうち、それらの領域の自由度の数である固定サイズの変数でなければなりません。接触等価節点法線力を表します。`multname_t` は、`multname_n` のサイズに `qdim-1` を掛けたサイズに対応する固定サイズの変数でなければなりません。これは接触等価節点接線（摩擦）力を表します。増大パラメータ `r` は、受容可能な値の範囲で選択されるべきです（弾性体の Young 率に近くなります、Getfem のユーザ文書を参照してください）。`friction_coeff` パラメータに格納された摩擦係数は、単一値または `multname_n` と同じサイズのベクトルです。オプションパラメータ `slave1` と `slave2` は、`rg1` と `rg2` で定義された領域が対応して“スレーブ”とみなされた場合に宣言します。デフォルトで `slave1` は真で、`slave2` は false です。すなわち `rg1` はスレーブ面を含み、`rg2` はマスター面を含みます。`slave1` と `slave2` のうちの 1 つだけが true に設定されるのが望ましいです。

パラメータ `aug_version` は、非対称の Alart-Curnier 拡張 Lagrangian の場合は 1、対称の場合は 2、拡張倍率を使用する非対称方法の場合は 3、拡張倍率および De Saxce 投影の場合の非対称方法の場合は増補戦略として 4 を示します。

基本的には、このブリックは行列 B_N と B_T とベクトル `gap` と `alpha` を計算し、基本的な接触要素を呼び出します。

22.24.14 摩擦のない接触状態を安定させるヒューズ

ヒューズにより安定化された摩擦のない接触要素を追加するには、`model` オブジェクトメソッドを呼び出します。

```
getfem::add_Hughes_stab_basic_contact_brick
    (md, varname_u, multname_n, dataname_r, BN, DN, dataname_gap, dataname_alpha, aug_version);
```

この関数は、乗数変数 `multname_n` のによりヒューズが安定した摩擦のない接触ブリックを `varname_u` に追加します。 U は片側拘束が適用される自由度のベクトルであり、 λ は接触力の乗数ベクトルです。そして、ヒューズは安定した摩擦のない接触条件が行列によって定義される B_N と D_N はこの条件が以下のように定義されていなければなりません。 $B_N U - D_N \lambda \leq 0$ 。ここで、 D_N は、安定化された項に対する質量行列です。変数 `dataname_r` は拡張パラメータの名前です r は許容可能な値の範囲で選択されるべきです。 `dataname_gap` は初期ギャップを表すオプションのパラメータです。これは、単一の値または値のベクトルにすることができません。 `dataname_alpha` は、拡張パラメータのためのオプションの均質化パラメータです。

パラメータ `aug_version` は、非対称の Alart-Curnier 拡張 Lagrange の場合は 1、対称の場合は 2、非対称の Alart-Curnier で拡張された乗数に基づく非対称の方法の場合は 3 です。

マトリクス D_N は、基本接触項とヒューズ安定項の和であることに留意してください。これは変更することができます

```
getfem::contact_brick_set_DN(md, indbrick);
```

22.24.15 剛性の障害物ブリックとの摩擦のない一体接触

```
getfem::add_integral_contact_with_rigid_obstacle_brick
    (md, mim, varname_u, multname_n, dataname_obs, dataname_r, region, option = 1);
```

この関数は、一体型の方法で定義されたモデルに剛性の障害物がある摩擦のない接触条件を追加します。これは、連続レベルで定義された拡張 Lagrangian 定式化の直接近似です。メリットはスケラビリティが向上していることです。Newton 反復の数は、メッシュサイズに多かれ少なかれ近くなければなりません。この条件は、`region` に対応する境界上の変数 `varname_u` に適用されます。剛体の障害物は、(有限要素法で補間された) 障害物までの符号付き距離であるデータ `dataname_obstacle` で記述されるべきです。 `multname_n` は接触応力を表

す有限要素法変数でなければなりません。multname_n と varname_u の間の極限条件が必要です。拡張パラメータ dataname_r は、許容可能な値の範囲で選択する必要があります。

オプションの可能な値は、非対称の Alart-Curnier 拡張 Lagrange 法の場合は 1、対称の場合は 2、追加の増強を伴う非対称の Alart-Curnier 法の場合は 3、新しい非対称法の場合は 4 です。デフォルト値は 1 です。

mim はもちろん積分方法を表します。それは、関係する非線形項を効率的に積分するのに十分正確でなければならないことに注意してください。

22.24.16 摩擦を伴う頑丈な障害物ブリックとの一体的な接触

```
getfem::add_integral_contact_with_rigid_obstacle_brick
  (md, mim, varname_u, multname_n, dataname_obs, dataname_r,
   dataname_friction_coeffs, region, option = 1, dataname_alpha = "",
   dataname_wt = "", dataname_gamma = "", dataname_vt = "");
```

この関数は、一体的に定義されたモデルに剛性の障害物を伴う摩擦状態の接触を追加します。これは、連続レベルで定義された拡張 Lagrangian 定式化の直接近似です。メリットはスケラビリティが向上していることです。Newton 反復の数は、メッシュサイズに多かれ少なかれ近くなければなりません。この条件は、region に対応する境界上の変数 varname_u に適用されます。剛体の障害物は、(有限要素法で補間された) 障害物までの符号付き距離であるデータ dataname_obstacle で記述されるべきです。multname_n は接触応力を表す有限要素法変数でなければなりません。multname_n と varname_u の間の極限条件が必要です。拡張パラメータ dataname_r は、許容可能な値の範囲で選択する必要があります。

パラメータ dataname_friction_coeffs には、Coulomb 摩擦係数とオプションとして接着剪断応力閾値と Tresca 限界せん断応力が含まれています。定数係数の場合、そのサイズは 1 から 3 までです。有限要素法で記述された係数の場合、このベクトルには対応する mesh_fem の基本自由度の数に等しい数の単一値、値のペアまたはトリプレットが含まれます。

オプションの可能な値は、非対称の Alart-Curnier 拡張 Lagrange 法の場合は 1、対称の場合は 2、追加の増強を伴う非対称の Alart-Curnier 法の場合は 3、新しい非対称法の場合は 4 です。デフォルト値は 1 です。オプション 4 は、純粋な Coulomb 摩擦を仮定し、接着応力とトレスカ限界係数を無視します。

dataname_alpha と dataname_wt は、進化的摩擦の問題を解決するためのオプションのパラメータです。dataname_gamma と dataname_vt は、摩擦条件にパラメータ依存滑り速度を加えるためのオプションのデータを示します。mim はもちろん積分方法を表します。それは、関係する非線形項を効率的に積分するのに十分正確でなければならないことに注意してください。

22.24.17 非整合メッシュ要素間の摩擦のない一体的接触

```
getfem::add_integral_contact_between_nonmatching_meshes_brick
  (md, mim, varname_u1, varname_u2, multname_n, dataname_r,
   region1, region2, option = 1);
```

この関数は、マッチしないメッシュ間の摩擦のない接触条件をモデルに追加します。これは、不可分な方法で定義されています。これは、連続レベルで定義された拡張 Lagrangian 定式化の直接近似です。メリットはスケーラビリティが向上している必要があります。Newton 反復の数は、メッシュサイズに多かれ少なかれなければなりません。条件は、region1 と region2 に対応する境界上の変数 varname_u1 と varname_u2 に適用されます。multname_n は接触応力を表す有限要素法変数でなければなりません。multname_n と varname_u1 と varname_u2 の間の極限条件が必要です。拡張パラメータ dataname_r は、許容可能な値の範囲で選択する必要があります。

オプションの可能な値は、非対称の Alart-Curnier 拡張 Lagrange 法の場合は 1、対称の場合は 2、追加の増強を伴う非対称の Alart-Curnier 法の場合は 3、新しい非対称法の場合は 4 です。デフォルト値は 1 です。

mim はもちろん積分方法を表します。それは、関係する非線形項を効率的に積分するのに十分正確でなければならないことに注意してください。

22.24.18 摩擦と非整合メッシュブリック間の一体的接触

```
getfem::add_integral_contact_between_nonmatching_meshes_brick
  (md, mim, varname_u1, varname_u2, multname, dataname_r,
   dataname_friction_coeffs, region1, region2, option = 1,
   dataname_alpha = "", dataname_wt1 = "", dataname_wt2 = "");
```

この関数は、一致していないメッシュ間の摩擦状態を持つ接点をモデルに追加します。この要素は、一体的に定義された接点を追加します。これは、連続レベルで定義された拡張 Lagrangian 定式化の直接近似です。メリットはスケーラビリティが向上している必要があります。Newton 反復の数は、メッシュサイズに多かれ少なかれなければなりません。条件は、region1 と region2 に対応する境界上の変数 varname_u1 と varname_u2 に適用されます。multname は、接触と摩擦のストレスを表す有限要素法変数でなければなりません。multname と varname_u1 と varname_u2 の間の極限条件が必要です。拡張パラメータ dataname_r は、許容可能な値の範囲で選択する必要があります。

パラメータ *dataname_friction_coeffs* には、Coulomb 摩擦係数とオプションとして接着剪断応力閾値と Treasca 限界せん断応力が含まれています。一定の係数の場合、そのサイズは 1 から 3 までです。varname_u1 と同じメッシュの有限要素法で記述された係数の場合、このベクトルには mesh_fem の基本的な自由度に対応する数値の数に等しい単一の値、値のペアまたはトリプレットが含まれます。

オプションの可能な値は、非対称の Alart-Curnier 拡張 Lagrange 法の場合は 1、対称の場合は 2、追加の増強を伴う非対称の Alart-Curnier 法の場合は 3、新しい非対称法の場合は 4 です。デフォルト値は 1 です。dataname_alpha

、 `dataname_wt1`、 `dataname_wt2` は、進化的摩擦の問題を解決するためのオプションのパラメータです。 `mim` は `varname_u1` と同じメッシュ上での積分法を表します。それは、関係する非線形項を効率的に積分するのに十分正確でなければならないことに注意してください。

22.24.19 剛性の障害物要素との摩擦のないペナルティ接触

```
getfem::add_penalized_contact_with_rigid_obstacle_brick
  (md, mim, varname_u, dataname_obs, dataname_r, region,
   option = 1, dataname_lambda_n = "");
```

この関数は、摩擦のないペナルティな接触状態と剛体の障害物をモデルに追加します。この条件は、 `region` に対応する境界上の変数 `varname_u` に適用されます。剛体の障害物は、(有限要素法で補間された) 障害物までの符号付き距離であるデータ `dataname_obstacle` で記述されるべきです。ペナルティ化パラメータ `dataname_r` は、近似非貫通条件を規定するのに十分な大きさですが、接線系の条件付けをあまりにも悪化させないように大きすぎないように選択すべきです。 `dataname_n` は、オプションが2の場合に使用されるオプションのパラメータです。この場合、ペナルティ項は `lambda_n` だけシフトされます(対応する拡張 **dLagrangian** の式に Uzawa アルゴリズムを使用できます)

22.24.20 摩擦を伴う頑丈な障害物要素とのペナルティ接触

```
getfem::add_penalized_contact_with_rigid_obstacle_brick
  (md, mim, varname_u, dataname_obs, dataname_r, dataname_friction_coeffs,
   region, option = 1, dataname_lambda = "", dataname_alpha = "",
   dataname_wt = "");
```

この関数は、Coulomb 摩擦を伴う不利な接触条件と、モデルに対する堅い障害物を追加します。この条件は、 `region` に対応する境界上の変数 `varname_u` に適用されます。剛性の障害物は、データ `dataname_obstacle` が障害物までの符号付き距離である(有限要素法で補間された) データで記述するべきです。

パラメータ `dataname_friction_coeffs` には、Coulomb 摩擦係数とオプションとして接着剪断応力閾値と Tresca 限界せん断応力が含まれています。定数係数の場合、そのサイズは1から3までです。有限要素法で記述された係数の場合、このベクトルには対応する `mesh_fem` の基本自由度の数に等しい数の単一値、値のペアまたはトリプレットが含まれます。

ペナルティ化パラメータ `dataname_r` は、近似的な非貫通および摩擦条件を規定するのに十分な大きさですが、接線系の条件をあまりにも悪化させないように大きすぎないように選択すべきです。 `dataname_lambda` は `option` が2の場合に使用されるオプションのパラメータです。この場合、ペナルティ項は `lambda` だけシフトします(対応する拡張 **Lagrangian** 式に対する Uzawa アルゴリズムの使用が可能になります)。 `dataname_alpha` と `dataname_wt` は、進化的摩擦の問題を解決するためのオプションのパラメータです。

22.24.21 非整合要素間の摩擦なしペナルティ接触

```
getfem::add_penalized_contact_between_nonmatching_meshes_brick
  (md, mim, varname_u1, varname_u2, dataname_r,
   region1, region2, option = 1, dataname_lambda_n = "");
```

この関数は、不整合メッシュ間にペナルティドコンタクト摩擦なし条件をモデルに追加します。この条件は、`region1` と `region2` に対応する境界上の変数 `varname_u1` と `varname_u2` に適用されます。ペナルティ化パラメータ `dataname_r` は、近似非貫通条件を規定するのに十分な大きさですが、接線系の条件付けをあまり大きく悪化させないように大きすぎないように選択すべきです。`dataname_n` は、オプションが 2 の場合に使用されるオプションのパラメータです。この場合、ペナルティ項は `lambda_n` だけシフトします（これにより対応する拡張 Lagrangian 式に Uzawa アルゴリズムを使用できます）

22.24.22 要素とマッチングしていないメッシュと摩擦との不利な接触

```
getfem::add_penalized_contact_between_nonmatching_meshes_brick
  (md, mim, varname_u1, varname_u2, dataname_r, dataname_friction_coeffs,
   region1, region2, option = 1, dataname_lambda = "",
   dataname_alpha = "", dataname_wt1 = "", dataname_wt2 = "");
```

この関数は、マッチングしていないメッシュ間の Coulomb 摩擦を含むペナルティ化された接触条件をモデルに追加します。この条件は、`region1` と `region2` に対応する境界上の変数 `varname_u1` と `varname_u2` に適用されます。ペナルティ化パラメータ `dataname_r` は、近似非貫通条件を規定するのに十分な大きさですが、接線系の条件付けをあまり大きく悪化させないように大きすぎないように選択すべきです。

パラメータ `dataname_friction_coeffs` には、Coulomb 摩擦係数とオプションとして接着剪断応力閾値と Tresca 限界せん断応力が含まれています。一定の係数については、そのサイズは 1 から 3 までです。`varname_u1` と同じメッシュ上の有限要素法で記述された係数の場合、このベクトルは、対応する `mesh_fem` の基本自由度数に等しいいくつかの単一値です。

`dataname_lambda` は `option` が 2 の場合に使用されるオプションのパラメータです。この場合、ペナルティ項は `lambda` だけシフトします（対応する拡張 Lagrangian 式に対する Uzawa アルゴリズムの使用が可能になります）`dataname_alpha`、`dataname_wt1`、`dataname_wt2` は、進化的摩擦の問題を解決するためのオプションのパラメータです。`mim` は `varname_u1` と同じメッシュ上での積分方法を表します。それは、関係する非線形項を効率的に積分するのに十分正確でなければならないことに注意してください。

22.25 摩擦ブリックとの有限すべり/有限変形接触

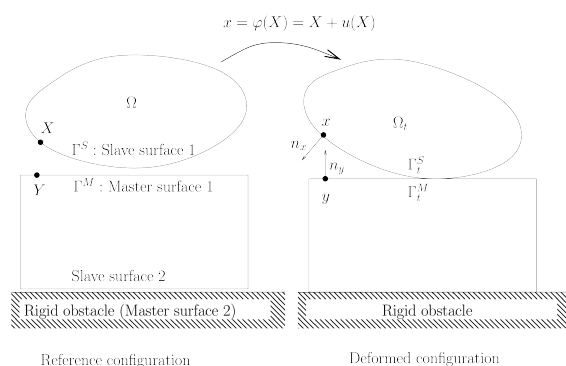
変形可能な構造の有限すべり/有限変形接触に対処するための基本的なツールは弱形式言語でアクセス可能です。いくつかの補間変換（補間変換を参照）は接触検出を実行し、反対側の境界境界にある接触境界から積分できる

ように定義されています。実際の構成における単位法線ベクトルや Coulomb 摩擦との接触を考慮する投影などの他の有用なツールも弱形式言語の演算子として定義されています。

もちろん、大スライド/有限変形接触アルゴリズムの計算コストは小スライド-小変形よりも大幅に高くなります。

22.25.1 レイトレーシング補間変換

高レベル汎用構築に接触検出を組み込むために、特定の補間変換が定義されています（内挿変換の詳細については補間変換を参照してください）。[KO-RE2014]に記載されているレイトレーシング接触検出に基づいており、以下で説明する基準を使用します。補間変換は、異なる可能性のある接触面を記憶します。ほとんどの方法で、潜在的接触面はマスターとスレーブの2つのカテゴリに分類されます（図を参照）。



スレーブ表面は“コンタクタ”で、マスターは“ターゲット”です。剛支障も考慮されます。それらは常にマスター表面です。基本的な規則は、スレーブ表面とマスター表面の間で接触が考慮されることです。しかし、マルチコンタクトフレームオブジェクトと *GetFEM++* 要素は2つのマスター表面間の接触、マスター表面と任意の数のスレーブ表面とマスター表面の自己接触を含む複数接触の状況を可能にします。

基本的には、接触対を検出するために、Gauss 点またはスレーブ表面の有限要素法節点はマスター表面に投影されます（図を参照）。自己接触が考慮される場合、Gauss 点またはマスター表面の有限要素法節点もマスター表面に投影されます。

レイトレーシング変換のモデルへの追加は次の通りです。

```
void add_raytracing_transformation(model &md, const std::string &transname,
                                   scalar_type d)
```

ここで `transname` は弱形式言語でそれを参照できる変換に与えられた名前、`d` は解放距離です（上記参照）。

レイトレーシング変換はスレーブまたはマスターの接触境界なしで追加されます。次の関数は変換にいくつかの境界を追加することを可能にします。

```
add_master_contact_boundary_to_raytracing_transformation(model &md,
    const std::string &transname, const mesh &m,
    const std::string &dispname, size_type region)
```

```
add_slave_contact_boundary_to_raytracing_transformation(model &md,
    const std::string &transname, const mesh &m,
    const std::string &dispname, size_type region)
```

ここで、`dispname` はその接触境界上の変位を表す変数名です。マスタとスレーブの接触境界の違いは、スレーブまたはマスタ境界からマスタ境界に向かって接触検出を実行することです。接触検出はスレーブ境界に向かって行われません。その結果、マスター表面の要素の影響ボックスのみが計算され格納されます。

次の関数で剛支障（マスター表面とみなされる）を追加することも可能です。

```
add_rigid_obstacle_to_raytracing_transformation(model &md,
    const std::string &transname,
    const std::string &expr, size_type N)
```

ここで、`expr` は弱形式言語の構文を使用した支障までの距離の記号表現です（ x は現在の位置、 $X(0)$ 、 $X(1)$... 対応する成分）。たとえば、式 $X(0) + 5$ は最初の座標の位置 -5 の右側にある平坦な障害物に対応します。式は符号付き距離に近いものでなければならないことに注意してください。特に、勾配ノルムは 1 に近づく必要があります。

非接触の状況と他の変形可能な物体との接触、または堅い障害物との接触を微分するために、変換は、弱形式言語の `Interpolate_filter` コマンドによって使用される整数識別を返します（[補間変換](#) を参照）。戻り値は次の通りです。

- 0: この Gauss 点で接点が見つかりません
- 1: この Gauss 点で変形可能な物体と接触します
- 2: この Gauss 点で剛体の障害物と接触します

これらの3つの場合、次のように微分することが可能です。


```
Interpolate_filter(transname, expr1, 0)
Interpolate_filter(transname, expr2, 1)
Interpolate_filter(transname, expr3, 2)
```

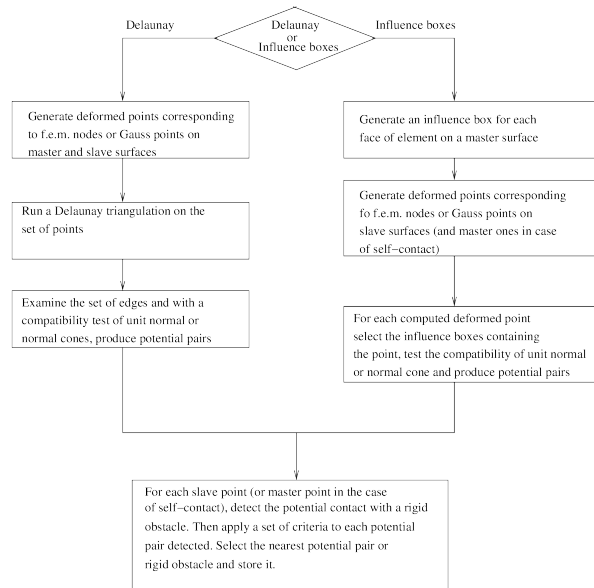
弱形式言語では、`expr1`、`expr2`、`expr3` は計算されるべき異なる項に対応しています。`matlab` インタフェースのデモプログラム `/interface/tests/matlab/demo_large_sliding_contact.m` は、使用例を示しています。


`model` オブジェクトが使用されていない場合、変換は `ga_workspace` オブジェクトで直接使用することもできます。詳細は、`getfem/getfem_contact_and_friction_common.h` を参照してください。また、`model`

オブジェクトのフレームワークでは、この変換のインターフェース使用が、以下に説明するモデルブリックによって許可されることにも注意してください。

22.25.2 接触対検出アルゴリズム

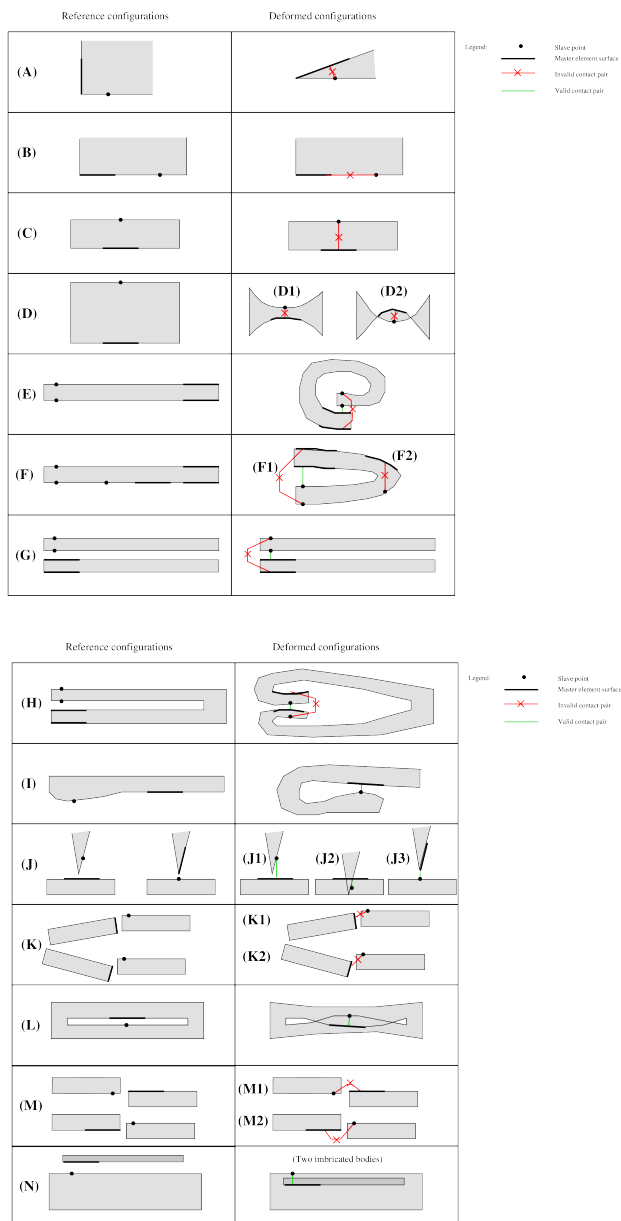
接触対は、スレーブ（または自己接触の場合はマスター）の点と、最も近いマスター表面（または剛性の障害物）の投影点とによって形成されます。使用されたアルゴリズムは  のように整理されます。



([Pantz2008] など、) グローバルなトポジカルな基準なしに有効な接触先と無効な接触先の間で確実に微分することは不可能です。しかし、この種の基準は、実施するのに非常にコストがかかる可能性があります。したがって、一般的には、可能性のあるすべてのケースをカバーすることができない簡単なヒューリスティックな基準を実装します。このような基準をここに提示します。これらはもちろん不完全であり、変更の対象となります。最初に、 では、条件を微分しなければならない有効なまたは無効な接触先の数の状況を見ることができます。

アルゴリズムの詳細は次の通りです。

- **影響ボックスの計算** 要素の影響ボックスは、放出距離に等しい距離でその境界ボックスのオフセットだけです。この戦略を使用する場合、リリース距離は要素サイズに比べて大きすぎるべきではありません。さもなければ、ポイントは接触対の検索をかなり減速させる多数の影響ボックスに対応します。インフルエンスボックスは、領域ツリーオブジェクトに格納され、平均複雑度を有するアルゴリズムを有する点を含むボックスを、 $O(\log(N))$ で見つけます。
- **潜在的な接触対とは何か。** 潜在的な接触対は調査されるスレーブポイント - マスターエレメント面のペアです。マスタ表面上のスレーブポイントの投影が行われ、基準が適用されます。




- **投影アルゴリズム。** マスター要素面へのスレーブ点の投影は、幾何変換と変位場による参照要素上の表面のパラメータ化によって行われます。投影中、要素面内にとどまる制約は適用されません。これは、要素面が解析的に延長されることを意味します。投影は、パラメータ化および Newton および/または BFGS アルゴリズムを使用して、スレーブ点と投影点との間の距離を最小にすることによって実行されます。raytrace が true に設定されている場合、投影は計算されません。その代わりに、点 x からの単位法線ベクトルの方向に y を見つけるためにレイトレーシングします。これは、通常の状態の逆を意味します (x は y の投影です)。

基準のリストは

- **基準 1** : ユニットの法線円錐/ベクトルは互換性があり、2つの点は同じ要素を共有しません。2つの単位法線ベクトルはスカラー積が非正であれば互換性があります。f.e.m の場合。フェムト節点は一般に複数の要素で共有されているため、各要素の単位法線ベクトルからなる法線円錐が考慮されます。少なくとも1組の単位法線ベクトルがスカラー積が非正であれば、2つのノーマルコーンは互換性があります。計算を単純化するために、ノーマルコーンの立体角がマルチコンタクトフレームオブジェクトのパラメータ `cut_angle` よりも小さい場合、ノーマルコーンは平均法線ベクトルに縮小されます。この基準は、問題 (B) および (K1) に対応することを可能にします。
- **基準 2** : 投影/レイトレースポイントの検索が収束しない場合、接触対は削除されます。マスターエレメント上のスレーブポイントの投影/レイトレースを計算するために使用された Newton 法 (および投影用の BFGS) 表面が収束しない場合、対は考慮されず警告が生成されます。
- **基準 3** : 投影されたポイントは要素の内側に設ける。スレーブポイントは、顔の内側に留まるという制約なしにマスター要素の表面に投影されます (つまり、面が長くなることを意味します)。正射影が顔の外側にある場合、ペアは考慮されません。しかし、現状では、問題 (J3) に対応するためには、随意的対応が考慮されなければなりません (この点では拘束を受けて顔に投影し、この時点で正常な円錐を試験する)。この基準は、(F2)、(K2)、(M1)、(M2) のケースに対応できます。
- **基準 4** : リリース距離が適用されます。スレーブポイントとマスター表面上の投影間の距離がリリース距離より大きい場合、接触対は考慮されません。リリース距離が適応され、変形がそれほど重要でない場合、(C)、(E)、(F1)、(G)、(H) の場合を処理できます。
- **判定基準 5** : 堅い障害物との比較 スレーブポイントとマスタ表面上の投影との間の符号付き距離が、堅い障害物を有するものより大きい場合 (リリース距離が剛性障害物にも最初に適用されることを考慮して) 接触対は考慮されません。
- **基準 6** : 自己接触のみ : 基準構成における単位法線の試験を適用します。自己接触の場合、スレーブ点とマスター要素が同じメッシュに属し、スレーブポイントはマスタ面の背後にあり (ユニットの外向き法線ベクトルに対して)、リリース距離の4倍も離れていません。これは、問題 (A)、(C)、(D)、(H) に対応することができます。
- **基準 7** : 接触対の最小符号付距離 保持された接触対 (または剛性障害物) の間で、符号付き最小距離に対応するものが保持されます。

投影による節点接触要素

表記 : $\Omega \subset \mathbb{R}^d$ は、変形可能な物体の参照構成を示し、いくつかの未接続部分 ( を参照) で構成されています。 Ω_t は変形後の構成で $\varphi^h : \Omega \rightarrow \Omega_t$ は有限要素法空間 V^h の変形近似です。変形 $u^h : \Omega \rightarrow \mathbb{R}^d$ は $\varphi^h(X) = X + u^h(X)$ により定義されます。参照構成 Ω の一般的な点は X によって表記されており、 $x = \varphi^h(X)$ によって変形後の対応する点が表記されています。変形後の構成の対応する境界はそれぞれ Γ_t^S と Γ_t^M です。(変形後構成の) 境界での点 $x = \varphi^h(X)$ における外向き単位基準化ベクトルは n_x により表現されています。最後に、表

記 $\delta A[B]$ は変形に関する量 A と方向 B の指向性導関数を表す。同様に、表記 $\delta^2 A[B, C]$ は方向 B と方向 C の 2 階導関数を示す。

$J(\varphi^h)$ は、接触と摩擦の寄与を考慮しない、システムのポテンシャルエネルギーです。典型的には、それは弾性及び外部負荷ポテンシャルエネルギーを含みます。 X_i は次のようにします。 $i \in I_{\text{nodes}}$ は参照設定のスレーブ境界上の有限要素節点の集合です。 $i \in I_{\text{def}}$ は、変形可能な物体のマスター表面と接触する可能性のある接触節点です。 X_i は次のようにします。 $i \in I_{\text{rig}}$ は、堅い障害物と接触する可能性のある接触節点です。

$x_i = \varphi^h(X_i)$ は、変形された構成の対応する節点であり、 y_i は変形された構成のマスター表面（または剛体の障害物）の投影です。 Y_i はマスター表面上点を確認する $y_i = \varphi^h(Y_i)$ です。これにより、通常のギャップを次のように定義できます。

$$g_i = n_y \cdot (\varphi^h(X_i) - \varphi^h(Y_i)) = \|\varphi^h(X_i) - \varphi^h(Y_i)\| \text{Sign}(n_y \cdot (\varphi^h(X_i) - \varphi^h(Y_i))),$$

ここで、 n_y は、 y のマスター表面の外向き単位法線ベクトルです。

定常剛性障害物のみを考慮し、Alart-Curnier Lagrangian [AL-CU1991] の原理を適用すると、摩擦条件との節点接触の問題は非対称バージョンで次のように表すことができる（線形弾性のケース）。

$$\begin{cases} \text{Find } \varphi^h \in V^h \text{ such that} \\ \delta J(\varphi^h)[\delta u^h] - \sum_{i \in I_{\text{def}}} \lambda_i \cdot (\delta u^h(X_i) - \delta u^h(Y_i)) - \sum_{i \in I_{\text{rig}}} \lambda_i \delta u^h(X_i) = 0 \quad \forall \delta u^h \in V^h, \\ \frac{1}{r} [\lambda_i + P_{n_y, \mathcal{F}}(\lambda_i + r(g_i n_y - \alpha(\varphi^h(X_i) - \varphi^h(Y_i)) - W_T(X_i) + W_T(Y_i)))] = 0 \quad \forall i \in I_{\text{def}}, \\ \frac{1}{r} [\lambda_i + P_{n_y, \mathcal{F}}(\lambda_i + r(g_i n_y - \alpha(\varphi^h(X_i) - W_T(X_i)))] = 0 \quad \forall i \in I_{\text{rig}}, \end{cases}$$

ここで、 $W_T, \alpha, P_{n_y, \mathcal{F}} \dots$ + 接線系

申し訳ありませんが、このブリックは動作していません。

22.25.3 摩擦接触のための高レベル汎用構築のツール

次の非線形演算子は、弱形式言語で定義されています（任意の項を計算する - 高水準の汎用的な構築手順 - 弱形式言語 参照）。

- Transformed_unit_vector(Grad_u, n) ここで Grad_u は変位フィールドの勾配であり、n は参照構成の単位ベクトルです。この非線形演算子は、以下の通りです。

$$n_{\text{trans}} = \frac{(I + \nabla u)^{-T} n}{\|(I + \nabla u)^{-T} n\|}$$

偏微分は以下の通りです。

$$\begin{aligned} \partial_u n_{\text{trans}}[\delta u] &= -(I - n_{\text{trans}} \otimes n_{\text{trans}})(I + \nabla u)^{-T} (\nabla \delta u)^T n_{\text{trans}} \\ \partial_n n_{\text{trans}}[\delta n] &= \frac{(I + \nabla u)^{-T} \delta n - n_{\text{trans}}(n_{\text{trans}} \cdot \delta n)}{\|(I + \nabla u)^{-T} n\|} \end{aligned}$$

- `Coulomb_friction_coupled_projection(lambda, n, Vs, g, f, r)` ここで `lambda` は接触力であり、`n` は単位法線ベクトルであり、`Vs` は摺動速度であり、`g` はギャップ、`f` は摩擦係数、`r` は正の増強パラメータです。演算子の式は次のとおりです。

$$P(\lambda, n, V_s, g, f, r) = -(\lambda \cdot n + rg)_- n + P_{B(n, \tau)}(\lambda - rV_s)$$

$$\text{with } \tau = \min(f_3 + f_1(\lambda \cdot n + rg)_-, f_2)$$

ここで、 $(\cdot)_-$ は負の部分 ($(x)_- = (-x)_+$) です。 f_1, f_2, f_3 は摩擦係数 3 成分です。成分 f_2, f_3 はオプションであることに注意してください。仮想係数が (f_1 のみ) 与えられている場合、これは古典 Coulomb 摩擦則に相当します。2 つの成分 (f_1, f_2 のみ) ベクトルが与えられている場合、これは与えられた閾値との Coulomb 摩擦に対応します。最後に、3 つの成分からなるベクトルが与えられた場合、摩擦法則は、上で与えられた τ の式に対応します。

式 $P_{B(n, \tau)}(q)$ は半径 τ の n に関して接線ボール上の直交射影 (これは [Return Mapping](#) 法へのリンクです) を指します。

微分は $T_n = (I - n \otimes n)$ および $q_T = T_n q$ のように表すことができます。

$$\partial_q P_{B(n, \tau)}(q) = \begin{cases} 0 & \text{for } \tau \leq 0 \\ \mathbf{T}_n & \text{for } \|q_T\| \leq \tau \\ \frac{\tau}{\|q_T\|} \left(\mathbf{T}_n - \frac{q_T}{\|q_T\|} \otimes \frac{q_T}{\|q_T\|} \right) & \text{otherwise} \end{cases}$$

$$\partial_\tau P_{B(n, \tau)}(q) = \begin{cases} 0 & \text{for } \tau \leq 0 \text{ or } \|q_T\| \leq \tau \\ \frac{q_T}{\|q_T\|} & \text{otherwise} \end{cases}$$

$$\partial_n P_{B(n, \tau)}(q) = \begin{cases} 0 & \text{for } \tau \leq 0 \\ -q \cdot n \mathbf{T}_n - n \otimes q_T & \text{for } \|q_T\| \leq \tau \\ -\frac{\tau}{\|q_T\|} \left(q \cdot n \left(\mathbf{T}_n - \frac{q_T}{\|q_T\|} \otimes \frac{q_T}{\|q_T\|} \right) + n \otimes q_T \right) & \text{otherwise.} \end{cases}$$

$$\partial_\lambda P(\lambda, n, V_s, g, f, r) = \partial_q P_{B(n, \tau)} + \partial_\tau P_{B(n, \tau)} \otimes \partial_\lambda \tau + H(-\lambda \cdot n - rg) n \otimes n,$$

$$\partial_n P(\lambda, n, V_s, g, f, r) = \left| \begin{array}{l} \partial_n P_{B(n, \tau)} + \partial_\tau P_{B(n, \tau)} \otimes \partial_n \tau \\ + H(-\lambda \cdot n - rg) (n \otimes \lambda - (2\lambda \cdot n + rg) n \otimes n + (\lambda \cdot n + rg) \mathbf{I}), \end{array} \right.$$

$$\partial_g P(\lambda, n, V_s, g, f, r) = \partial_\tau P_{B(n, \tau)} \partial_g \tau + H(-\lambda \cdot n - rg) r n$$

$$\partial_f P(\lambda, n, V_s, g, f, r) = \partial_\tau P_{B(n, \tau)} \partial_f \tau$$

$$\partial_r P(\lambda, n, V_s, g, f, r) = H(-\lambda \cdot n - rg) gn + \partial_q P_{B(n, \tau)} V_s + \partial_\tau P_{B(n, \tau)} \partial_r \tau$$

22.25.4 レイトレース付き積分接触要素

ブリックを追加します。

```
indbrick = add_integral_large_sliding_contact_brick_raytracing
(model &md, const std::string &dataname_r,
 scalar_type release_distance,
 const std::string &dataname_friction_coeff = "0",
 const std::string &dataname_alpha = "1");
```

このブリックは、複数の接触先の状況に対処できます。前のセクションで説明したように、モデルにレイトレーシング補間変換を追加します。その名前はコマンドによって取得できます。

```
const std::string &transformation_name_of_large_sliding_contact_brick(model &md,
    size_type indbrick);
```

ブリックがモデルに追加されると、マスターとスレーブの接触境界を次の関数で追加する必要があります。

```
add_contact_boundary_to_large_sliding_contact_brick(model &md,
    size_type indbrick, const mesh_im &mim, size_type region,
    bool is_master, bool is_slave, const std::string &u,
    const std::string &lambda = "", const std::string &w = "",
    bool frame_indifferent = false)
```

ここで、`region` は境界を表す有効なメッシュ領域番号でなければなりません。`is_master` はその接触境界で接触検出が行われるなら `true` にセットされ、`is_slave` はその境界上で接触項の積分が行われる場合は `true` に設定してください。接触境界は、特に自己接触検出を可能にするために、マスターとスレーブの両方になることが可能であることに留意してください。`u` は変位変数です。`is_slave` が `true` に設定されている場合、`lambda` は接触境界上の自由度を持つ乗数変数を記述しなければなりません（通常は `md.add_filtered_fem_variable(...)` メソッドでモデルに追加されます）。純粋なマスター接触境界は、乗数の定義を必要としません。さらに、`w` は進展する場合であり、前の時間ステップにおける変位を表します。

剛性の障害物を要素に加えることができます:

```
add_rigid_obstacle_to_large_sliding_contact_brick(model &md,
    size_type indbrick, std::string expr, size_type N)
```

ここで、`expr` は弱形式言語（`X` は現在の位置）を使った式で、障害物までの符号付き距離でなければなりません。`N` はメッシュ寸法です。

第 23 章

数値連続法と分岐

FEM モデルの離散化から生じる代数的問題を、以下の形式で書くことができます

$$F(U) = 0.$$

以下の説明では、モデルが追加のスカラーパラメータ λ に依存していると仮定します。そのため $F(U) = F(U, \lambda)$ です。

23.1 数値連続法

数値連続法は、システムの解を追跡するために役立ちます

$$F(U, \lambda) = 0, \quad F: \mathbb{R}^N \times \mathbb{R} \rightarrow \mathbb{R}^N.$$

GetFEM++ には、区分的 C^1 (PC^1) の解曲線の連続手法が実装されています (詳細は [Li-Re2014] を参照)。状態変数 U とパラメータ λ との間に明示的な違いがないので、簡潔にするため、 $Y := (U, \lambda)$ と表記します。併も、例えば、接線を計算する際のスケールリングが悪くなるのを避けるために、以下の加重スカラー積とノルムを使用します。

$$\langle Y, \tilde{Y} \rangle_w := \kappa \langle U, \tilde{U} \rangle + \lambda \tilde{\lambda}, \quad \|Y\|_w := \sqrt{\kappa \|U\|^2 + \lambda^2}, \quad Y = (U, \lambda), \quad \tilde{Y} = (\tilde{U}, \tilde{\lambda}).$$

ここで、 κ は $\kappa \langle U, \tilde{U} \rangle$ が対応する空間変数のスカラー積に比例するように選ぶ必要があり、通常 L^2 です。たとえば、次のようにすることができます $\kappa = h^d$ 、ここで、 h はメッシュサイズで、 d は対称の問題の次元を表します。あるいは、 κ は、簡単にするために $1/N$ として設定することもできます。

連続法の考え方は、古典的な予測子修正子法によって滑らかな解曲線を追跡し、滑らかな部分を連続的に結合することです。

使用される特定の予測子修正子法は、MATCONT [Dh-Go-Ku2003] に実装されている 不正確な *Moore-Penrose* 連続をわずかに変更して使用します。これは、解曲線上におおよそ対応する単位接線ベクトル T_j のシーケンスであ

る連続する点 Y_j の連続を計算します :

$$\|F(Y_j)\| \leq \varepsilon, \quad F'(Y_j; T_j) = 0, \quad \|T_j\|_w = 1, \quad j = 0, 1, \dots$$

それを説明するために、上記の処理の関係を満たすカップル (Y_j, T_j) を考えます。予測では、次のような (Y_{j+1}, T_{j+1}) の初期近似が成り立ちます。

$$Y_{j+1}^0 := Y_j + h_j T_j, \quad T_{j+1}^0 := T_j,$$

ここで、 h_j はステップサイズです。その選択については後で説明します。

補正では、次のようなシーケンスが計算されます $\{(Y_{j+1}^l, T_{j+1}^l)\}$, ここで $T_{j+1}^l := \tilde{T}_{j+1}^l / \|\tilde{T}_{j+1}^l\|_w$ とカップル $(Y_{j+1}^l, \tilde{T}_{j+1}^l)$ は方程式 $F^l(Y, T) = 0$ に適用される Newton 法のイテレーションにより与えられます。

$$F^l(Y, T) := \begin{pmatrix} F(Y) \\ (T_{j+1}^{l-1})^\top (Y - Y_{j+1}^{l-1}) \\ \nabla F(Y_{j+1}^{l-1}) T \\ \langle T_{j+1}^{l-1}, T \rangle_w - \langle T_{j+1}^{l-1}, T_{j+1}^{l-1} \rangle_w \end{pmatrix}$$

初期近似 $(Y_{j+1}^{l-1}, T_{j+1}^{l-1})$ は次のようになります。 F のポテンシャルの非微分可能性により、Newton 法の微分的に滑らかな変形が使用されます ([Fa-Pa2003] のアルゴリズム 7.2.14)。

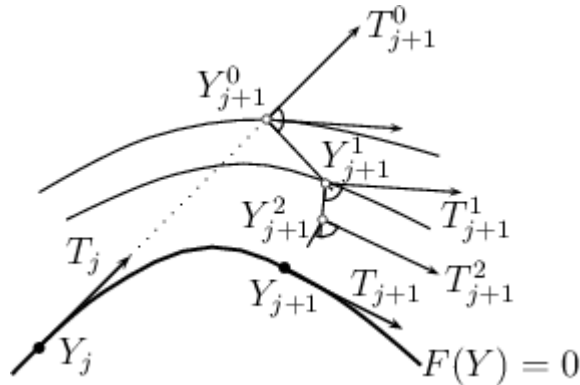


図 23.1 補正。

カップル (Y_{j+1}^l, T_{j+1}^l) は (Y_{j+1}, T_{j+1}) を満たす、もし $\|F(Y_{j+1}^l)\| \leq \varepsilon$ と $\|Y_{j+1}^l - Y_{j+1}^{l-1}\|_w \leq \varepsilon'$, と T_{j+1}^l と T_j の間の角度の余弦は c_{\min} のものより大きいか等しい F (または、非微分可能性の場合にはその選択関数の 1 つ) の部分勾配が、解析的に組み立てられているのに対して、 λ は $1e-8$ に等しい増分で前方有限差分法によって評価されます。

次の予測におけるステップサイズ h_{j+1} は、Newton 補正がどのように成功したかに依存します。次の式で必要な反復回数 l_{it} を示します。

$$h_{j+1} := \begin{cases} \max\{h_{dec} h_j, h_{\min}\} & \text{if no new couple has been accepted,} \\ \min\{h_{inc} h_j, h_{\max}\} & \text{if a new couple has been accepted and } l_{it} < l_{thr}, \\ h_j & \text{otherwise,} \end{cases}$$

ここで、 $0 < h_{dec} < 1 < h_{inc}$, $0 < l_{thr}$ と $0 < h_{min} < h_{max}$ は定数となります。最初は、いくつかの $h_{min} \leq h_{init} \leq h_{max}$ に対して $h_1 := h_{init}$ のように設定します。

ここで、滑らかな振る舞いの 1 つのサブ領域に対応する解曲線の一部を近似したとします。 F の滑らかな振る舞いの別のサブ領域に対応する部分を回復したいと考えます。 (Y_j, T_j) は最後に計算されたカップルとします。

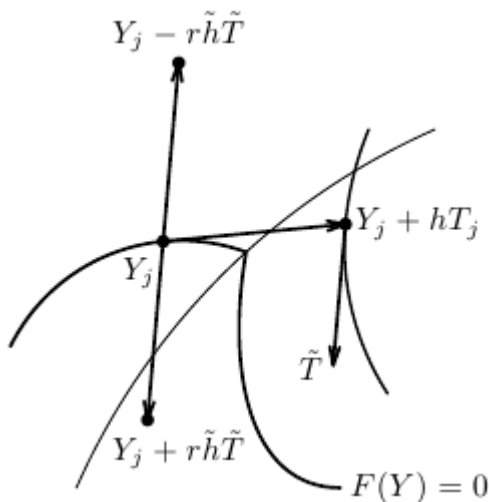


図 23.2 解曲線の滑らかな区分間の遷移。

他の滑らかな区分の接線を近似するには、まず、次のような点 $Y_j + hT_j$ をとります。ここで、 h は h_{min} よりも少し大きいです、そのためこの点が円滑な動作の他のサブ領域の内部に属するようにします。それから、 \tilde{T} は以下となります

$$\nabla F(Y_j + hT_j)\tilde{T} = 0, \quad \|\tilde{T}\|_w = 1,$$

そしてこれはこのベクトルの適切な方向を決定するために残っています。これは、以下の観察に基づいて行うことができます。第 1 に、 $Y_j - r\tilde{h}\tilde{T}$ が Y_j の任意の正の \tilde{h} と同じサブドメイン内にあるような $r \in \{\pm 1\}$ が存在します。これは、 $\nabla F(Y_j - r\tilde{h}\tilde{T})T_- = 0$ が $\frac{|T_-^\top \tilde{T}|}{\|T_-\| \|\tilde{T}\|}$ が 1 よりもかなり小さいという事実によって特徴づけられます。第 2 に、 $Y_j + r\tilde{h}\tilde{T}$ は、ある正の閾値よりも大きい \tilde{h} のために、他のサブ領域に現れ、このような値の場合、 $\frac{|T_+^\top \tilde{T}|}{\|T_+\| \|\tilde{T}\|}$ は T_+ が $\nabla F(Y_j + r\tilde{h}\tilde{T})T_+ = 0$ の場合に 1 に近いです。

これは、 \tilde{T} : の所望の方向を選択するための以下の手順を示唆しています: \tilde{h} の値を h_{min} から連続的に増加させ、 \tilde{h} と $r \in \{\pm 1\}$ に到着したときに

$$\frac{|T^\top \tilde{T}|}{\|T\| \|\tilde{T}\|} \approx 1 \quad \text{if } \nabla F(Y_j + r\tilde{h}\tilde{T})T = 0,$$

他の滑らかな区分の接線の近似値として $r\tilde{T}$ を取ります。

この近似を使って、予測子修正子を $(Y_j, r\tilde{T})$ のように再始動します。

GetFEM++ では、モデルパラメータ化の 2 つの連続の方法が実装されています。

1. パラメータ λ は、モデルが依存するスカラーデータです。
2. このモデルはモデルが依存するベクトルデータ P によって スカラーパラメータ λ によりパラメータ化されます。この場合、線形経路を使用します

$$\lambda \mapsto P(\lambda) := (1 - \lambda)P^0 + \lambda P^1,$$

ここで、 P^0 と P^1 には、 P の値が与えられ、問題の解の集合をトレースします。

$$F(U, P(\lambda)) = 0.$$

23.2 限界点の検出

システム $F(U, \lambda) = 0$ の解を追跡する場合、**限界点**（折り返し点または折り返し点とも呼ばれます）に注目します、これにより同じ値に対する解の数 λ は変わります。これらの点は、試行関数 τ_{LP} の符号変化によって検出することができます

$$\tau_{LP}(T_j)\tau_{LP}(T_{j+1}) < 0,$$

ここで、 τ_{LP} は次のように定義されます。

$$\tau_{LP}(T) := T_\lambda, \quad T = (T_U, T_\lambda) \in \mathbb{R}^N \times \mathbb{R}.$$

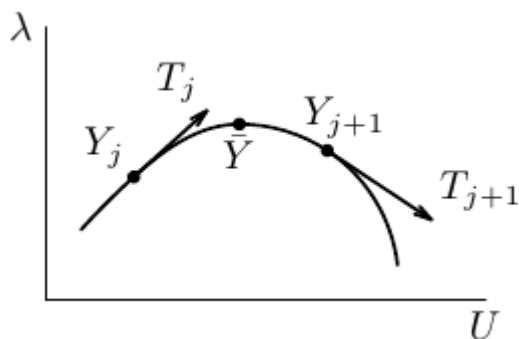


図 23.3 限界点。

23.3 数値分岐

点 \bar{Y} はシステム $F(Y) = 0$ の分岐点と呼ばれます $F(\bar{Y}) = 0$ ならば、2つ以上の異なる解曲線がそれを通過します。次の結果は、滑らかな分岐点のテストを与えます（例えば、[Georg2001]を参照）。

$s \mapsto Y(s)$ は解曲線のパラメータ化であり、 $\bar{Y} := Y(\bar{s})$ は分岐点です。さらに、 $T^\top \dot{Y}(\bar{s}) > 0$ 、 $B \notin \text{Im}(J(\bar{Y}))$ 、 $C \notin \text{Im}(J(\bar{Y})^\top)$ 、 $d \in \mathbb{R}$ として、

$$J(Y) := \begin{pmatrix} \nabla F(Y) \\ T^\top \end{pmatrix}.$$

以下で $\tau_{\text{BP}}(Y)$ を定義します

$$\begin{pmatrix} J(Y) & B \\ C^\top & d \end{pmatrix} \begin{pmatrix} V(Y) \\ \tau_{\text{BP}}(Y) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

次に $\tau_{\text{BP}}(Y(s))$ は、 $s = \bar{s}$ で記号を変換します。

明らかに、 B 、 C 、 d をランダムに取る場合、それらが上記の要件を満たす可能性が高くなります。したがって、数値連続法は、各連続ステップでの補正によって提供されたベクトル Y と T を取り、次の τ_{BP} の符号を監視することによって分岐点を検出することができます。

分岐点 \bar{Y} が $\tau_{\text{BP}}(Y_j)\tau_{\text{BP}}(Y_{j+1}) < 0$ の符号の変化によって検出されると、これは、特別なステップ長適応 ([Al-Ge1997] の 8.1 節を参照) を用いて上記の予測子修正子のステップによってより正確に近似することができます。すなわち、次のステップ長を

$$h_{j+1} := -\frac{\tau_{\text{BP}}(Y_{j+1})}{\tau_{\text{BP}}(Y_{j+1}) - \tau_{\text{BP}}(Y_j)} h_j$$

$|h_{j+1}| < h_{\min}$ のようになるまで、関数 $s \mapsto \tau_{\text{BP}}(Y(s))$ のゼロを見つけるための割線のメソッドに対応させます。

最後に、解の分岐を切り替えたいとしましょう。この目的のために、我々は、2つの異なる解曲線が交差する、いわゆる **単純な分岐点** の場合を考えます。

\tilde{Y} は、与えられた \bar{Y} の近似であり、 $V(\tilde{Y})$ は、試行関数 $\tau_{\text{BP}}(\tilde{Y})$ を計算するための拡張システムの解の最初の部分です。[Georg2001] で提案されているように、 $V(\tilde{Y})$ を予測変数として使用し、 $(\tilde{Y}, V(\tilde{Y}))$ から始まる1つの連続ステップを実行して、新しい分岐上の点を取得できます。この連続ステップが正常に実行され、新しい分岐上の点がりカバリされた後、通常の予測子修正子ステップを続行して、この分岐をトレースすることができます。

最近では、 PC^1 -分岐のための数値計算ツールが *GetFEM++* で開発されました。 J は、次の式で定義される実数パラメータの行列関数となります。

$$J(\alpha) := (1 - \alpha) \begin{pmatrix} \nabla F(Y_j) \\ T_j^\top \end{pmatrix} + \alpha \begin{pmatrix} \nabla F(Y_{j+1}) \\ T_{j+1}^\top \end{pmatrix}.$$

[Li-Re2014hal] で提案されているように、以下のテストは、 Y_j と Y_{j+1} の間の分岐点 PC^1 の検出に使用できます。

$$\det J(0) \det J(1) < 0.$$

このテストを数値的に実行するために、以下を紹介します。

$$M(\alpha) := \begin{pmatrix} J(\alpha) & B \\ C^\top & d \end{pmatrix}$$

と $\tau_{BP}(\alpha)$ は上記と同様に

$$M(\alpha) \begin{pmatrix} V(\alpha) \\ \tau_{BP}(\alpha) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

これは、Cramer 法から、

$$\tau_{BP}(\alpha) = \frac{\det J(\alpha)}{\det M(\alpha)}$$

ただし、 $\det M(\alpha)$ はゼロではありません。したがって、 $\det J(\alpha)$ がゼロのときに $\det M(\alpha)$ が 0 でないように B 、 C および d が選択された場合、 $\det J(\alpha)$ の符号変更は $\tau_{BP}(\alpha)$ から 0 の通過によって特徴付けられ、 $\det M(\alpha)$ の符号は $\tau_{BP}(\alpha)$ の符号変化によります。 $\det M(\alpha)$ の符号変化が特異点に起因して変化することによります。結論として、 $\det J(0) \det J(1)$ の符号は α が $[0, 1]$ を通過するときの $\tau_{BP}(\alpha)$ の挙動を追跡し、 $\det J(\alpha)$ の符号変化を監視することによって決定されます。

[Li-Re2014hal] で示されているように、 B 、 C 、 d はランダムに選択することができます。現在の値 α のインクリメント δ は適宜変更され、 τ_{BP} の特異点が有効に扱われます。 $\tau_{BP}(\alpha)$ のそれぞれの計算の後、 δ は次のように設定されます。

$$\delta := \begin{cases} \min\{2\delta, \delta_{\max}\} & \text{if } |\tau_{BP}(\alpha) - \tau_{BP}(\alpha - \delta)| < 0.5\tau_{\text{fac}}\tau_{\text{ref}}, \\ \max\{0.1\delta, \delta_{\min}\} & \text{if } |\tau_{BP}(\alpha) - \tau_{BP}(\alpha - \delta)| > \tau_{\text{fac}}\tau_{\text{ref}}, \\ \delta & \text{otherwise,} \end{cases}$$

ここで、 $\delta_{\max} > \delta_{\min} > 0$ で、 $\tau_{\text{fac}} > 0$ は既定の定数で、 $\tau_{\text{ref}} := \max\{|\tau_{BP}(1) - \tau_{BP}(0)|, 10^{-8}\}$ です。

Y_j と Y_{j+1} の間で分岐点 PC^1 が検出されると、それは二分法のような手順でより正確に近似されます。得られた近似は、 Y_j と同じ滑らかな枝上にあり、滑らかさの対応する領域から対応する単位接線も計算されます。

前です (例えば、`getfem::default_linear_solver<getfem::model_real_sparse_matrix, getfem::model_real_plain_vector>(model)`)。実数 h_{init} 、 h_{max} 、 h_{min} 、 h_{inc} 、 h_{dec} は h_{init} 、 h_{max} 、 h_{min} 、 h_{inc} 、および h_{dec} を表し、整数 maxit 、 thrit 、 maxres 、 maxdiff 、 mincos 、 maxres_solve は l_{thr} 、 ε 、 ε' 、 c_{min} であり、線形システムの目標残差値をそれぞれ求めるために設定されます。非負の整数 noisy は、連続プロセスの過程でどの程度詳細な情報が表示するかを決定します (詳細なほど値が大きくなります)。整数 singularities は、特異点を指定する必要があります (完全に無視する場合は 0、限界点を検出する場合は 1、分岐点を検出して処理する場合は 2)。プール値 nosmooth は、滑らかな連続法か分岐かを指定する必要があります。また、滑らかでないツールでも手法を指定する必要があります。実数 delta_max 、 delta_min 、および thrvar は、 δ_{max} 、 δ_{min} と τ_{fac} を表します。 ndir と nspan は n_{dir} と n_{span} をそれぞれ意味します。

必要に応じて、ベクトルデータによるパラメータ化は、次のように宣言されます。

```
S.set_parametrised_data_names(initdata_name, finaldata_name, currentdata_name);
```

ここで、データ名 initdata_name と finaldata_name は、それぞれ、 P^0 と P^1 を表す必要があります。 currentdata_name の下には、 $P(\lambda)$ の値が格納されていなければなりません。すなわち、モデルが依存するデータの実際の値です。

次に、連続法は次の式で初期化されます

```
S.init_Moore_Penrose_continuation(U, lambda, T_U, T_lambda, h);
```

ここで、 U は $Y_0 = (U, \lambda)$ におけるパラメータの値の解でなければなりません。この初期化の間、初期値は T_{lambda} の符号に従って計算され、 Y_0 に対応する初期単位正接 T_0 が返され、 T_U 、 T_{lambda} が返されます。さらに、 h は初期ステップサイズ h_{init} に設定されます。

その後、連続の際の 1 ステップは次のように呼び出されます

```
S.Moore_Penrose_continuation(U, lambda, T_U, T_lambda, h, h0);
```

各呼び出しの後、解曲線上の新しい点および対応する接線が、変数 U 、 λ および T_U 、 T_{lambda} で返されます。次の予測のステップサイズは h で返されます。現在のステップのサイズはオプションの引数 h_0 に返されます。選択された特異点の値によれば、各連続ステップの終わりに、限界点および分岐点の試行関数が評価されます。さらに、滑らかな分岐点が検出されれば、数値分岐の手順が実行され、分岐点の近似および両方の分岐曲線に対する接線が連続オブジェクト S に保存されます。そこから、次のいずれかのカーブを連続的にトレースするため初期化できるように、 S のメンバ関数で簡単に回復することができます。

平滑問題を使用する例の完全な例はテストプログラム `tests/test_continuation.cc`、`interface/tests/matlab/demo_continuation.m` と `interface/src/scilab/demos/demo_continuation.s` にあります、一方 `interface/src/scilab/demos/demo_continuation_vee.sce` と `interface/src/scilab/demos/demo_continuation_block.sce` は平滑でない例になります。

第 24 章

有限歪弾性ブリック

このブリックは、大きな変形弾性のためのいくつかの古典的な超弾性構成則を実装します。

24.1 有限歪み弾性に関するいくつかの復習

Ω を参照構成とし、 Ω_t を弾性体の変形構成とします。そして、 $X \in \Omega$ に対して次のように変形を表します $\Phi(x) = u(X) + X$ 。ベクトルフィールド u は、初期位置に対する変位です。

Cauchy-Green テンソルは次のように定義されます。

$$C = \nabla\Phi^T \nabla\Phi$$

変形勾配テンソル (Green-Lagrange)

$$E = \frac{1}{2} (\nabla\Phi^T \nabla\Phi - I) = \frac{1}{2} (\nabla u^T \nabla u + \nabla u^T + \nabla u)$$

(線形弾性の場合、 $\nabla u^T \nabla u$ は無視されます)。

ここで

$$C = \nabla\Phi^T \nabla\Phi = 2E + I.$$

2つのテンソル E と C は、有限ひずみ弾性構成則を記述するために使用されます。

24.1.1 主要な不変量および導関数

有限ひずみ弾性構成則の記述は、しばしば、変形テンソルの主要な不変量を必要とします：

i_1, i_2, i_3 は次数 1, 2 と 3 の不変量です :

$$\begin{aligned} i_1(E) &= \text{tr } E & i_1(C) &= 2\text{tr } E + 3 \\ i_2(E) &= \frac{(\text{tr } E)^2 - \text{tr } E^2}{2} & i_2(C) &= 4i_2(E) + 4i_1(E) + 3 \\ i_3(E) &= \det E & i_3(C) &= 8i_3(E) + 4i_2(E) + 2i_1(E) + 1 \end{aligned}$$

H 方向のテンソル E の不変量の導関数 H は次のとおりです。

$$\begin{aligned} \frac{\partial i_1}{\partial E}(E; H) &= I : H = \text{tr } H \\ \frac{\partial i_2}{\partial E}(E; H) &= (i_1(E)I - E^T) : H = (\text{tr } E)(\text{tr } H) - E^T : H \\ \frac{\partial i_3}{\partial E}(E; H) &= i_3(E)(E^{-T}) : H = (i_2(E)I - i_1(E)E + E^2) : H \text{ in } 3D. \end{aligned}$$

ゆえに

$$\begin{aligned} \frac{\partial i_1}{\partial E}(E) &= I \\ \frac{\partial i_2}{\partial E}(E) &= i_1(E)I - E^T \\ \frac{\partial i_3}{\partial E}(E) &= i_3(E)E^{-T}. \end{aligned}$$

また再掲すると

$$\frac{\partial(M^{-1})}{\partial M}(M; H) = -M^{-1}HM^{-1}$$

不変量の 2 次導関数は、次の式によって定義される 4 次テンソルとなります。

$$\begin{aligned} \frac{\partial^2 i_1}{\partial E^2}(E) &= 0 \\ \frac{\partial^2 i_2}{\partial E^2}(E)_{ijkl} &= \delta_{ij}\delta_{kl} - \delta_{il}\delta_{jk} \\ \frac{\partial^2 i_3}{\partial E^2}(E)_{ijkl} &= i_3(E)(E_{ji}^{-1}E_{lk}^{-1} - E_{jk}^{-1}E_{li}^{-1}). \end{aligned}$$

表記法 $A : B$ は Frobenius 積を表します。この積 $A : B = \sum_{ij} A_{ij}B_{ij}$ には以下の特性があります :

$$\begin{aligned} A : B &= \text{tr } (A^T B) = \text{tr } (AB^T) = \text{tr } (BA^T) = \text{tr } (B^T A), \\ A : BC &= B^T A : C, \\ A : BC &= AC^T : B, \\ \text{tr } (ABC) &= \text{tr } (B^T A^T C^T) \end{aligned}$$

また、

$$\frac{\partial i_j}{\partial E}(C; H) = 2 \frac{\partial i_j}{\partial C}(C; H).$$

この性質により、Cauchy-Green テンソル不変量の関数として、特に汎用化された Blatz-Ko ひずみエネルギーの場合は、構成的規則を表現することが可能になります。

24.1.2 弾性ポテンシャルエネルギーとその導関数

基準構成における応力は、第 2 種 Piola-Kirchhoff 応力テンソル $\hat{\sigma} = \nabla\Phi^{-1}\sigma\nabla\Phi^{-t} \det \nabla\Phi$ によって記述することができます。ここで、 σ は、変形された構成 Ω_t の Cauchy 応力テンソルです。超弾性の構成則は次のように与えられます。

$$\hat{\sigma} = \frac{\partial}{\partial E} W(E) = 2 \frac{\partial}{\partial C} W(C)$$

ここで、 W は材料のひずみエネルギーの密度です。全ひずみエネルギーは次式となります。

$$\mathcal{I}(u) = \int_{\Omega} W(E(u)) dX$$

また、 v 方向のエネルギーの導関数は次のように書くことができます。

$$D\mathcal{I}(u; v) = \int_{\Omega} \frac{\partial W}{\partial E}(E(u)) : (I + \nabla u^T) \nabla v dX$$

また特に

$$\begin{aligned} DE(u; v) &= \frac{1}{2} (\nabla u^T \nabla v + \nabla v^T \nabla u + \nabla v^T + \nabla v) \\ &= \frac{1}{2} (\nabla v^T (I + \nabla u) + (I + \nabla u^T) \nabla v) \end{aligned}$$

A が対称であるとき、 $\hat{\sigma}$ の場合には、 $A : B = A : (B + B^T) / 2$ となります。

別の方法として、静的平衡を考慮するものがあり、これは参照構成において以下のように書くことができます。

$$-\operatorname{div} \left((I + \nabla u) \hat{\sigma} \right) = f.$$

部分積分により、次のようになります。

$$\int_{\Omega} (I + \nabla u) \hat{\sigma} : \nabla v dX = l(v).$$

24.1.3 接線行列

変位 u は固定されています。接線行列を得るためには、 u を $u + h$ に置き換えます。

$$\int_{\Omega} (I + \nabla u + \nabla h) \hat{\sigma}(E(u) + E(h) + \frac{1}{2} (\nabla h^T \nabla u + \nabla u^T \nabla h)) : \nabla v dX = l(v)$$

線形部分に関して h を考慮します。ここで

$$\begin{aligned} & \int_{\Omega} \nabla h \hat{\sigma}(E(u)) : \nabla v dX + \\ & \int_{\Omega} \frac{\partial^2 W}{\partial E^2} \left(\frac{\nabla h + \nabla h^T + \nabla h^T \nabla u + \nabla u^T \nabla h}{2} \right) : (I + \nabla u^T) \nabla v dX \end{aligned}$$

それは対称に関して v と h であり。これは次のように書き換えることができます。

$$\int_{\Omega} \nabla h \hat{\sigma}(E(u)) : \nabla v + \mathcal{A}((I + \nabla u^T) \nabla h) : (I + \nabla u^T) \nabla v \, dX$$

ここで \mathcal{A} は対称であり $3 \times 3 \times 3 \times 3$ テンソルは $\mathcal{A}_{ijkl} = ((\frac{\partial^2 W}{\partial E^2})_{ijkl} + (\frac{\partial^2 W}{\partial E^2})_{ijlk})/2$ により与えられます。

24.1.4 いくつかの古典的な構成則

線形化された : **Saint-Venant Kirchhoff** 則 (微小変形)

$$\begin{aligned} W &= \frac{\lambda}{2} i_1(E)^2 + \mu i_1(E^2) \\ \hat{\sigma} &= \lambda i_1(E) I + 2\mu E \\ \mathcal{A} &= \lambda i_1(H) I + \mu(H + H^T) \end{aligned}$$

3つのパラメータ **Mooney-Rivlin** 法

圧縮可能な材料です。

$$W = c_1(j_1(C) - 3) + c_2(j_2(C) - 3) + d_1(i_3(C)^{1/2} - 1)^2$$

ここで c_1, c_2 と d_1 は係数が与えられ、

$$\begin{aligned} j_1(C) &= i_1(C) i_3(C)^{-1/3} \\ j_2(C) &= i_2(C) i_3(C)^{-2/3} \\ \frac{\partial j_1}{\partial C}(C) &= i_3(C)^{-1/3} \left(\frac{\partial i_1}{\partial C}(C) - \frac{i_1(C)}{3i_3(C)} \frac{\partial i_3}{\partial C}(C) \right) \\ \frac{\partial j_2}{\partial C}(C) &= i_3(C)^{-2/3} \left(\frac{\partial i_2}{\partial C}(C) - \frac{2i_2(C)}{3i_3(C)} \frac{\partial i_3}{\partial C}(C) \right) \\ \frac{\partial^2 j_1}{\partial C^2}(C) &= i_3(C)^{-1/3} \left(\frac{4i_1(C)}{9i_3(C)^2} \frac{\partial i_3}{\partial C}(C) \otimes \frac{\partial i_3}{\partial C}(C) - \frac{1}{3i_3(C)} \left(\frac{\partial i_3}{\partial C}(C) \otimes \frac{\partial i_1}{\partial C}(C) \right. \right. \\ &\quad \left. \left. + \frac{\partial i_1}{\partial C}(C) \otimes \frac{\partial i_3}{\partial C}(C) \right) - \frac{i_1(C)}{3i_3(C)} \frac{\partial^2 i_3}{\partial C^2}(C) \right) \\ \frac{\partial^2 j_2}{\partial C^2}(C) &= i_3(C)^{-2/3} \left(\frac{\partial^2 i_2}{\partial C^2}(C) + \frac{10i_2(C)}{9i_3(C)^2} \frac{\partial i_3}{\partial C}(C) \otimes \frac{\partial i_3}{\partial C}(C) \right. \\ &\quad \left. - \frac{2}{3i_3(C)} \left(\frac{\partial i_3}{\partial C}(C) \otimes \frac{\partial i_2}{\partial C}(C) + \frac{\partial i_2}{\partial C}(C) \otimes \frac{\partial i_3}{\partial C}(C) \right) - \frac{2i_2(C)}{3i_3(C)} \frac{\partial^2 i_3}{\partial C^2}(C) \right) \end{aligned}$$

その後

$$\begin{aligned}\hat{\sigma} &= 2c_1 \frac{\partial j_1}{\partial C}(C) + 2c_2 \frac{\partial j_2}{\partial C}(C) + 2d_1 \left(1 - i_3(C)^{-1/2}\right) \frac{\partial i_3}{\partial C}(C) \\ \mathcal{B} &= 4c_1 \frac{\partial^2 j_1}{\partial C^2}(C) + 4c_2 \frac{\partial^2 j_2}{\partial C^2}(C) + 4d_1 \left(\left(1 - i_3(C)^{-1/2}\right) \frac{\partial^2 i_3}{\partial C^2}(C) + \frac{1}{2} i_3(C)^{-3/2} \frac{\partial i_3}{\partial C}(C) \otimes \frac{\partial i_3}{\partial C}(C) \right) \\ \mathcal{A}_{ijkl} &= (\mathcal{B}_{ijkl} + \mathcal{B}_{jikl})/2\end{aligned}$$

非圧縮性の材料です。

$$d_1 = 0$$

with the additional constraint:

$$i_3(C) = 1$$

非圧縮性制約 $i_3(C) = 1$ は Lagrange 乗数 p (圧力) で扱われます。

$$\text{拘束: } \sigma = -pI \Rightarrow \hat{\sigma} = -p\nabla\Phi\nabla\Phi^{-T} \det \nabla\Phi$$

$$\begin{aligned}1 - i_3(\nabla\Phi) &= 0 \\ - \int_{\Omega_0} (\det \nabla\Phi - 1) q dX &= 0 \quad \forall q\end{aligned}$$

$$\begin{aligned}B &= - \int_{\Omega_0} p(\nabla\Phi)^{-T} \det \nabla\Phi : \nabla v dX \\ K &= \int_{\Omega_0} (p(\nabla\Phi)^{-T} (\nabla h)^T (\nabla\Phi)^{-T} \det \nabla\Phi : \nabla v dX - p(\nabla\Phi)^{-T} (\det \nabla\Phi (\nabla\Phi)^{-T} : \nabla h) : \nabla v) dX \\ &= \int_{\Omega_0} p(\nabla h^T \nabla\Phi^{-T}) : (\nabla\Phi^{-1} \nabla v) \det \nabla\Phi dX - \int_{\Omega_0} p(\nabla\Phi^{-T} : \nabla h)(\nabla\Phi^{-T} : \nabla v) \det \nabla\Phi dX\end{aligned}$$

Ciarlet-Geymonat 法

$$W = a i_1(C) + \left(\frac{\mu}{2} - a\right) i_2(C) + \left(\frac{\lambda}{4} - \frac{\mu}{2} + a\right) i_3(C) - \left(\frac{\mu}{2} + \frac{\lambda}{4}\right) \log \det(C)$$

Lame 係数 λ, μ は $\max(0, \frac{\mu}{2} - \frac{\lambda}{4}) < a < \frac{\mu}{2}$ のようになります。 ([ciarlet1988] を参照してください)。

一般化された **Blatz-Ko 法**

$$W = (ai_1(C) + bi_3(C)^{1/2} + c \frac{\mathbb{B}_2(C)}{\mathbb{B}_3(C)} + d)^n$$

$\frac{\partial}{\partial C} W(C) = \sum_j \frac{\partial W}{\partial i_j(C)} \frac{\partial i_j(C)}{\partial C}$ と $\frac{\partial^2}{\partial C^2} W(C) = \sum_j \sum_k \frac{\partial^2 W}{\partial i_j(C) \partial i_k(C)} \frac{\partial i_k(C)}{\partial C} \otimes \frac{\partial i_j(C)}{\partial C} + \sum_j \frac{\partial W}{\partial i_j(C)} \frac{\partial^2 i_j(C)}{\partial C^2}$ であるため、Cauchy-Green テンソル不変量に対して歪エネルギー関数の導関数を計算しなければなりません ($\frac{\partial i_j}{\partial E}(C; H) = 2 \frac{\partial i_j}{\partial C}(C; H)$ であるため、以下の点 E に関して不変量導関数を計算する必要はありません。)

$$\begin{aligned} \frac{\partial W}{\partial i_1(C)} &= naZ^{n-1} \quad \text{with } Z = (ai_1(C) + bi_3(C)^{1/2} + c\frac{b_2(C)}{b_3(C)} + d) \\ \frac{\partial W}{\partial i_2(C)} &= n\frac{c}{i_3(C)} Z^{n-1} \\ \frac{\partial W}{\partial i_3(C)} &= n\left(\frac{b}{2i_3(C)^{1/2}} - \frac{ci_2(C)}{i_3(C)^2}\right) Z^{n-1} \\ \frac{\partial^2 W}{\partial i_1^2(C)} &= n(n-1)A^2 Z^{n-2} \\ \frac{\partial^2 W}{\partial i_1(C) \partial i_2(C)} &= n(n-1)A\frac{c}{i_3(C)} Z^{n-2} \\ \frac{\partial^2 W}{\partial i_1(C) \partial i_3(C)} &= n(n-1)A\left(\frac{b}{2i_3(C)^{1/2}} - \frac{ci_2(C)}{i_3(C)^2}\right) Z^{n-2} \\ \frac{\partial^2 W}{\partial i_2^2(C)} &= n(n-1)\frac{c^2}{i_3(C)^2} Z^{n-2} \\ \frac{\partial^2 W}{\partial i_2(C) \partial i_3(C)} &= n(n-1)\left(\frac{b}{2i_3(C)^{1/2}} - \frac{ci_2(C)}{i_3(C)^2}\right) Z^{n-2} - n\frac{c^2}{i_3(C)^2} Z^{n-1} \\ \frac{\partial^2 W}{\partial i_3^2(C)} &= n(n-1)\left(\frac{b}{2i_3(C)^{1/2}} - \frac{ci_2(C)}{i_3(C)^2}\right)^2 Z^{n-2} + n\left(-\frac{b}{4i_3(C)^{3/2}} + 2\frac{ci_2(C)}{i_3(C)^4}\right) Z^{n-1} \end{aligned}$$

平面ひずみ超弾性

以上のすべてのモデルは、ボリウム領域で有効です。対応する平面歪み 2D モデルは、応力テンソルと 4 次テンソル A をその平面成分に制限することで得られます。

24.2 モデルに非線形弾力性ブリックを追加する

この要素は大ひずみ弾性問題を表しています。ファイル `getfem/getfem_nonlinear_elasticity.h` と `getfem/getfem_nonlinear_elasticity.cc` で定義されています。このブリックをモデルに追加する関数は以下の通りです。

```
ind = getfem::add_nonlinear_elasticity_brick
      (md, mim, varname, AHL, dataname, region = -1);
```

AHL は、考えられる超弾性法則を表す `getfem::abstract_hyperelastic_law` 型のオブジェクトです。次の中から選択する必要があります。

```
getfem::SaintVenant_Kirchhoff_hyperelastic_law AHL;
getfem::Ciarlet_Geymonat_hyperelastic_law AHL;
getfem::Mooney_Rivlin_hyperelastic_law AHL(compressible, neohookean);
getfem::plane_strain_hyperelastic_law AHL(pAHL);
getfem::generalized_Blatz_Ko_hyperelastic_law AHL;
```

Saint-Venant Kirchhoff 則は、2 つのラメ係数で定義された線形化された法則であり、Ciarlet Geymonat 則は 2 つのラメ係数と追加の係数 (λ, μ, a) で定義されます。

Mooney-Rivlin 則は、2つのオプションのフラグを受け取ります。最初のもは、材料が圧縮可能かどうかを判定し ($d_1 \neq 0$)、2番目のフラグは材料が Neo-Hookean ($c_2 = 0$) かどうかを判定します。これらのフラグに応じて、1~3個の係数が必要な場合があります。デフォルトでは、それは非圧縮性かつ Neo-Hookean でないと定義されているため、2つの物質係数 (c_1, c_2) が必要です。この場合、大きなひずみの非圧縮条件で使用する必要があります。

平面ひずみ超弾性則は、パラメータとして超弾性則の指針をとり、2D 平面ひずみ近似を実行します。

md はモデル変数、mim は積分方法、varname は文字列に変数が追加される変数の名前、dataname は法則の係数を表すモデル内のデータの名称です (定数または有限要素法を記述することができる) また region はその項が考慮される領域です (デフォルトではすべてのメッシュ)。

tests ディレクトリ内のプログラム nonlinear_elastostatic.cc と interface/tests/matlab ディレクトリ内のプログラム demo_nonlinear_elasticity.m はこの要素の非圧縮性条件での使用の例です。

新しい超弾性構成則の追加は、歪エネルギー、応力テンソルおよび応力テンソルの導関数の表現を与えることに留意してください。詳細は、ファイル getfem/getfem_nonlinear_elasticity.cc を参照してください。特に、不変量およびそれらの導関数の表現が利用可能です。

Von Mises または Tresca の応力を計算する関数も利用できます。

```
VM = compute_Von_Mises_or_Tresca
    (md, varname, AHL, dataname, mf_vm, VM, tresca)
```

これは、有限要素法 mf_vm 上の Von Mises または Tresca 応力の自由度のベクトルを返します。tresca はブール値で、Tresca 応力に対しては true、Von Mises 応力には false となります。

24.3 モデルに大ひずみの非圧縮性ブリックを追加する

このブリックは、大きなひずみ問題で非圧縮性条件を追加します

$$\det(I + \nabla u) = 1,$$

圧力を表す Lagrange 乗数が混合式で導入されます。このブリックをモデルに追加する関数は次の通りです。

```
ind = add_nonlinear_incompressibility_brick
    (md, mim, varname, multaname, region = -1)
```

ここで、md はモデル、mim は積分法、varname は非圧縮性条件が追加されたモデルの変数、multaname は圧力に対応する乗数変数 (変数の有限要素法と乗数の間で少なくとも線形の Ladyzhenskaja-Babuska-Brezzi 極限条件が満たされていることに注意してください。) region は、項が考慮されるメッシュ領域に対応するオプションのパラメータです (デフォルトではすべてのメッシュ)。

24.4 高レベル汎用構築バージョン

弱形式言葉は、*GetFEM++* で実装された超弾性ポテンシャルおよび構成規則にアクセスできるようにします。これにより、言語で直接使用することができます。たとえば、モデル内の汎用的な構築ブリックを使用するか、または特定の数量の補間（応力など）が可能です。

非線形弾性に役立つ非線形演算子の言語のリストです:

```
Det (M) % determinant of the matrix M
Trace (M) % trace of the matrix M
Matrix_i2 (M) % second invariant of M (in 3D): (sqr(Trace(m)) - Trace(m*m))/2
Matrix_j1 (M) % modified first invariant of M: Trace(m)pow(Det(m),-1/3).
Matrix_j2 (M) % modified second invariant of M: Matrix_I2(m)*pow(Det(m),-2/3)
Right_Cauchy_Green (F) % F' * F
Left_Cauchy_Green (F) % F * F'
Green_Lagrangian (F) % (F'F - Id(meshdim))/2
Cauchy_stress_from_PK2 (sigma, Grad_u) % (Id+Grad_u)*sigma*(I+Grad_u')/det(I+Grad_u)
```

ポテンシャルのリストです:

```
Saint_Venant_Kirchhoff_potential (Grad_u, [lambda; mu])
Plane_Strain_Saint_Venant_Kirchhoff_potential (Grad_u, [lambda; mu])
Generalized_Blatz_Ko_potential (Grad_u, [a;b;c;d;n])
Plane_Strain_Generalized_Blatz_Ko_potential (Grad_u, [a;b;c;d;n])
Ciarlet_Geymonat_potential (Grad_u, [lambda;mu;a])
Plane_Strain_Ciarlet_Geymonat_potential (Grad_u, [lambda;mu;a])
Incompressible_Mooney_Rivlin_potential (Grad_u, [c1;c2])
Plane_Strain_Incompressible_Mooney_Rivlin_potential (Grad_u, [c1;c2])
Compressible_Mooney_Rivlin_potential (Grad_u, [c1;c2;d1])
Plane_Strain_Compressible_Mooney_Rivlin_potential (Grad_u, [c1;c2;d1])
Incompressible_Neo_Hookean_potential (Grad_u, [c1])
Plane_Strain_Incompressible_Neo_Hookean_potential (Grad_u, [c1])
Compressible_Neo_Hookean_potential (Grad_u, [c1;d1])
Plane_Strain_Compressible_Neo_Hookean_potential (Grad_u, [c1;d1])
Compressible_Neo_Hookean_Bonet_potential (Grad_u, [lambda;mu])
Plane_Strain_Compressible_Neo_Hookean_Bonet_potential (Grad_u, [lambda;mu])
Compressible_Neo_Hookean_Ciarlet_potential (Grad_u, [lambda;mu])
Plane_Strain_Compressible_Neo_Hookean_Ciarlet_potential (Grad_u, [lambda;mu])
```

第2種 Piola-Kirchhoff 応力テンソルです:

```
Saint_Venant_Kirchhoff_sigma (Grad_u, [lambda; mu])
Plane_Strain_Saint_Venant_Kirchhoff_sigma (Grad_u, [lambda; mu])
Generalized_Blatz_Ko_sigma (Grad_u, [a;b;c;d;n])
Plane_Strain_Generalized_Blatz_Ko_sigma (Grad_u, [a;b;c;d;n])
Ciarlet_Geymonat_sigma (Grad_u, [lambda;mu;a])
Plane_Strain_Ciarlet_Geymonat_sigma (Grad_u, [lambda;mu;a])
```

```

Incompressible_Mooney_Rivlin_sigma(Grad_u, [c1;c2])
Plane_Strain_Incompressible_Mooney_Rivlin_sigma(Grad_u, [c1;c2])
Compressible_Mooney_Rivlin_sigma(Grad_u, [c1;c2;d1])
Plane_Strain_Compressible_Mooney_Rivlin_sigma(Grad_u, [c1;c2;d1])
Incompressible_Neo_Hookean_sigma(Grad_u, [c1])
Plane_Strain_Incompressible_Neo_Hookean_sigma(Grad_u, [c1])
Compressible_Neo_Hookean_sigma(Grad_u, [c1;d1])
Plane_Strain_Compressible_Neo_Hookean_sigma(Grad_u, [c1;d1])
Compressible_Neo_Hookean_Bonet_sigma(Grad_u, [lambda;mu])
Plane_Strain_Compressible_Neo_Hookean_Bonet_sigma(Grad_u, [lambda;mu])
Compressible_Neo_Hookean_Ciarlet_sigma(Grad_u, [lambda;mu])
Plane_Strain_Compressible_Neo_Hookean_Ciarlet_sigma(Grad_u, [lambda;mu])

```

材料パラメータに関する導関数は、Saint Venant Kirchhoff 超弾性法のためには個別に実装されていないことに注意してください。したがって、パラメータをモデルの他の変数に依存させることはできません（導関数は実装するのはそれほど複雑ではありませんが、現時点では古い実装のラッパーのみが記述されています）。

モデルの結合は、弱形式レベルで行われることに注意してください。汎用的な方法では、ポテンシャルで問題を定義しないことをお勧めします。第 1 に、ポテンシャルの 2 次導関数（接線系を得るために必要）が非常に複雑で最適化不可能であり、第 2 に、ポテンシャルレベルで主結合を得ることができないという 2 つの理由があります。従って、ポテンシャルの使用は、ポテンシャルの実際の計算に限定されるべきです。

モデルまたは `ga_workspace` の変数 `u` に Saint Venant-Kirchhoff 超弾性項を追加するには、次の構築文字列を使用します。

```
"((Id(meshdim)+Grad_u)*(Saint_Venant_Kirchhoff_sigma(Grad_u,[lambda;mu]))):Grad_Test_u"
```

その場合、データ `lambda` と `mu` を `model/ga_workspace` で宣言する必要があることに注意してください。もちろん、いくつかのデータに応じて陽な定数や式で置き換えることも可能です。

非圧縮性の Mooney-Rivlin 則に関しては、非圧縮性の項を完成されなければなりません。たとえば、次の非圧縮ブリックについて、

```
ind = add_finite_strain_incompressibility_brick(md, mim, varname, multname, region = -1);
```

このブリックは、 p が乗数で u 変位を表す変数であれば項 $p \cdot (1 - \text{Det}(\text{Id}(\text{meshdim}) + \text{Grad}_u))$ を与える単純なものです。

モデルに超弾性項を加えることは、次の関数によっても行うことができます:

```
ind = add_finite_strain_elasticity_brick(md, mim, lawname, varname, params,
                                         region = size_type(-1));
```

ここで、`md` はモデル、`mim` は積分法、`varname` は大きなひずみ変位を表すモデルの変数、`lawname` は構成則名で `Saint_Venant_Kirchhoff`、`Generalized_Blatz_Ko`、

Ciarlet_Geymonat、 Incompressible_Mooney_Rivlin、 Compressible_Mooney_Rivlin、 Incompressible_Neo_Hookean、 Compressible_Neo_Hookean、 Compressible_Neo_Hookean_Bonet または Compressible_Neo_Hookean_Ciarlet のうちのどれかです。params は小ベクトルまたはベクトルフィールドとして定義された構成則のパラメータを表す文字列です。

Von Mises 応力は次の関数で補間することができます

```
void compute_finite_strain_elasticity_Von_Mises(md, varname, lawname, params, mf_vm, VM,
                                             rg=mesh_region::all_convexes());
```

ここで、md はモデル、varname は大ひずみ変位を表すモデルの変数、lawname は構成則名（前述のブリックを参照）、params は文字列補間が行われる法則のパラメータ、mf_vm は（不連続なのが好ましい）Lagrange 有限要素法を表現し、VM は補間された値が格納される model_real_plain_vector 型のベクトルを表します。

第 25 章

微小ひずみの可塑性

GetFEM++ における可塑性モデルの近似のフレームワークについて説明します。ブリックを実装し新しい可塑性モデルを拡張する場合は `interface/src/gf_model_set.cc` と `interface/src/gf_model_set.cc` を参照してください。

25.1 理論的背景

微小ひずみの可塑性について簡単に紹介します。より詳細な説明については、主に [SI-HU1998] および [SO-PE-OW2008] を参照してください。

25.1.1 微小ひずみテンソルの加算分解

$\Omega \subset \mathbb{R}^3$ は変形可能な物体の参照構成であり、 $u : \Omega \rightarrow \mathbb{R}^3$ は変形フィールドです。微小ひずみの可塑性は、微小ひずみテンソル $\varepsilon(u) = \frac{\nabla u + \nabla u^T}{2}$ の加法分解に基づいています。

$$\varepsilon(u) = \varepsilon^e + \varepsilon^p$$

ここで、 ε^e はひずみテンソルの弾性部分であり、 ε^p は弾性部分です。

25.1.2 内部変数、自由エネルギーポテンシャル、弾性法則

以下を考慮します

$$\alpha : \Omega \rightarrow \mathbb{R}^{d_\alpha},$$

これはひずみ型の内部変数 d_α のベクトルフィールドです。(内部変数が考慮されていない場合は、 $d_\alpha = 0$ となります。) 自由エネルギーポテンシャルも考慮します。

$$\psi(\varepsilon^e, \alpha),$$

対応する応力型変数は以下で決定されます。

$$\sigma = \frac{\partial \psi}{\partial \varepsilon^e}(\varepsilon^e, \alpha), \quad A = \frac{\partial \psi}{\partial \alpha}(\varepsilon^e, \alpha),$$

ここで、 σ は Cauchy の応力テンソルであり、 A は応力型の内部変数です。降伏条件は

$$\sigma : \dot{\varepsilon}^p - A : \dot{\alpha} \geq 0.$$

通常、 $\psi(\varepsilon^e, \alpha)$ は次のように分解されます。

$$\psi(\varepsilon^e, \alpha) = \psi^e(\varepsilon^e) + \psi^p(\alpha).$$

線形化弾性体の場合、 $\psi^e(\varepsilon^e) = \frac{1}{2}(\mathcal{A}\varepsilon^e) : \varepsilon^e$ は 4 次の弾性テンソルです。等方性線形弾性の場合、この式は次のようになります。 $\psi^e(\varepsilon^e) = \mu \text{dev}(\varepsilon^e) : \text{dev}(\varepsilon^e) + \frac{1}{2}K(\text{tr}(\varepsilon^e))^2$ ここで、 μ はせん断弾性率で $K = \lambda + 2\mu/3$ は Bulk 弾性率です。

25.1.3 塑性ポテンシャル、降伏関数、塑性流れ則

応力が臨界値に達すると、塑性降伏が起これと考えられます。この限界は、降伏関数 $f(\sigma, A)$ と条件によって決定されます。

$$f(\sigma, A) \leq 0.$$

表面 $f(\sigma, A) = 0$ は、塑性変形が起こる降伏面です。

また、(両方の変数に関して凸包である) 塑性ポテンシャル $\Psi(\sigma, A)$ を考慮してみましょう。塑性の流れ則と方向は次のように定義されます。

$$\dot{\varepsilon}^p = \gamma \frac{\partial \Psi}{\partial \sigma}(\sigma, A), \quad \dot{\alpha} = -\gamma \frac{\partial \Psi}{\partial A}(\sigma, A),$$

付加的な相補性条件として以下が成り立ちます。

$$f(\sigma, A) \leq 0, \quad \gamma \geq 0, \quad f(\sigma, A)\gamma = 0.$$

変数 γ は塑性乗数と呼ばれます。以下の場合に注意してください。 $\psi(\varepsilon^e, \alpha), f(\sigma, A)$ と $\Psi(\sigma, A)$ が微分可能でないとき、副差を使用しなければなりません。関連する可塑性は $\Psi(\sigma, A) = f(\sigma, A)$ に対応します。

25.1.4 初期境界値問題

動的弾塑性問題の弱定式化は、任意の運動学的に許容される次のような試行関数 v で書くことができます。

$$\begin{cases} \int_{\Omega} \rho \ddot{u} \cdot v + \sigma : \nabla v dx = \int_{\Omega} f \dot{v} dx + \int_{\Gamma_N} g \dot{v} dx, \\ u(0, x) = u_0(x), \quad \dot{u}(0) = v_0(x), \\ \varepsilon^p(0, x) = \varepsilon_0^p, \quad \alpha(0, x) = \alpha_0, \end{cases}$$

$u_0, v_0, \varepsilon_0^p, \alpha_0$ は初期値で、 f と g は領域の内部で、 Ω と境界の部分には Γ_N があります。

可塑性モデルは、しばしば、 $\rho \ddot{u}$ 項が無視される準静的問題に適用されることに注意してください。

時間ステップ $\Delta t = t_{n+1} - t_n$ が与えられたとします、続いて時刻 t_n から t_{n+1} までの、 u_n, ε_n^p と α_n 時刻 t_n の $u(t_n), \varepsilon_n^p(t_n)$ と $\alpha(t_n)$ の近似について示します。選択された時間積分スキームの近似は（例えば、提案されたスキームの1つ **過渡問題の積分のためのモデルツール**）流れ則の積分に使用される時間積分スキームとは異なっても構いません（以下を参照）。

25.2 流れ則の積分

塑性流れ則は、独自の時間積分法で積分する必要があります。標準法の中で、後方 Euler 法では、 θ 法（または汎用された台形ルール）と汎用された中間点法が、この文脈で最も一般的に使用されます。ここでは、 θ -法（ $\theta = 1$ は特殊なケースとして後方 Euler 法に対応します）を選択します。

u_{n+1} は、考慮する時間ステップの変位であり、前のステップの変位は u_n です。

塑性流れ則の積分のための θ -法は

$$\varepsilon_{n+1}^p - \varepsilon_n^p = (1 - \theta) \Delta t \gamma_n \frac{\partial \Psi}{\partial \sigma}(\sigma_n, A_n) + \theta \Delta t \gamma_{n+1} \frac{\partial \Psi}{\partial \sigma}(\sigma_{n+1}, A_{n+1}), \quad (25.1)$$

$$\alpha_{n+1} - \alpha_n = -(1 - \theta) \Delta t \gamma_n \frac{\partial \Psi}{\partial A}(\sigma_n, A_n) - \theta \Delta t \gamma_{n+1} \frac{\partial \Psi}{\partial A}(\sigma_{n+1}, A_{n+1}), \quad (25.2)$$

相補性条件は

$$f(\sigma_{n+1}, A_{n+1}) \leq 0, \quad \gamma_{n+1} \geq 0, \quad f(\sigma_{n+1}, A_{n+1}) \gamma_{n+1} = 0.$$

ここで、 $0 < \theta \leq 1$ は θ -法のパラメータです。 $\theta = 0$ は、可塑性の陽な積分を考慮しないため除外します。 $\theta = 1$ は後方 Euler 法に対応し、 $\theta = 1/2$ は2次整合法である Crank-Nicolson（または台形法）法に対応します。時間ステップ n における量の相補性条件は、前の時間ステップにより処理されています（ σ_n, α_n , と γ_n は、すでに決定しています）。

解はすべての未知数、すなわち次のような全体の問題すべてを解く必要があります。それは $u_{n+1}, \gamma_{n+1}, \varepsilon_{n+1}^p$ and A_{n+1} です。これはもちろん可能ですが、結果として高い自由度がとなるため、かなり高コストな戦略になります。古典的な戦略（たとえば、[SO-PE-OW2008] または最も近い 1 点投影法を参照）は、考慮する積分法の各 Gauss 点上の塑性流動を別々に、またより正確に写像します。

$$\begin{aligned}\mathcal{E}^p &: (u_{n+1}, \zeta_n, \eta_n) \mapsto \varepsilon_{n+1}^p \\ \mathcal{A} &: (u_{n+1}, \zeta_n, \eta_n) \mapsto \alpha_{n+1}\end{aligned}$$

η_n, ζ_n により方程式 (25.1), (25.2) の右辺は次のようになります。

$$\begin{aligned}\zeta_n &= \varepsilon_n^p + (1 - \theta)\Delta t \gamma_n \frac{\partial \Psi}{\partial \sigma}(\sigma_n, A_n), \\ \eta_n &= \alpha_n - (1 - \theta)\Delta t \gamma_n \frac{\partial \Psi}{\partial A}(\sigma_n, A_n)\end{aligned}$$

これは、特に次のことを意味します ($\varepsilon_{n+1}^p, \alpha_{n+1}$) = ($\mathcal{E}^p(u_{n+1}, \zeta_n, \eta_n), \mathcal{A}(u_{n+1}, \zeta_n, \eta_n)$) は方程式 (25.1) と (25.2) の解です。これらの写像とそれらの正接モジュラス（通常、一貫した正接モジュラスと呼ばれる）は、Newton 法を用いた問題のグローバルな解法で使用され、 u_{n+1} は一意に残りの変数です。Return Mapping 法の利点は、グローバル解の一意の変数が変位 u_{n+1} であることです。各 Gauss 点上の非線形解法がしばしば必要であり、通常はローカル Newton 法で実行される。

GetFEM++ では我々は Return Mapping 法と、主に [PO-NI2016]、[SE-PO-WO2015] および [HA-WO2009] に触発された、より簡単な接線モジュラスを可能にする以下に展開される代替戦略の両方を提案します。それは u_{n+1} に関して、さらに未知数なものとして、 γ_{n+1} （倍の数）の値を保存する必要があります。それからわかるように、これは、降伏関数のより汎用的な処理を可能にします。これは、この追加の未知数のスカラー場の単純さと引き換えとなっています。

まず、後ほど現れる（単純なローカル逆変換が可能になる） $\alpha(\sigma_{n+1}, A_{n+1}) > 0$ という追加の（そしてオプションの）関数と新しい未知のスカラー場を考えます

$$\xi_{n+1} = \frac{\gamma_{n+1}}{\alpha(\sigma_{n+1}, A_{n+1})},$$

私たちの 2 つの主要な未知数は u_{n+1} と ξ_{n+1} になりました。離散化された塑性流れ則の積分は、次のようになります。

$$\varepsilon_{n+1}^p - \varepsilon_n^p = (1 - \theta)\alpha(\sigma_n, A_n)\Delta t \xi_n \frac{\partial \Psi}{\partial \sigma}(\sigma_n, A_n) + \theta\alpha(\sigma_{n+1}, A_{n+1})\Delta t \xi_{n+1} \frac{\partial \Psi}{\partial \sigma}(\sigma_{n+1}, A_{n+1}), \quad (25.3)$$

$$\alpha_{n+1} - \alpha_n = (1 - \theta)\alpha(\sigma_n, A_n)\Delta t \xi_n \frac{\partial \Psi}{\partial A}(\sigma_n, A_n) + \theta\alpha(\sigma_{n+1}, A_{n+1})\Delta t \xi_{n+1} \frac{\partial \Psi}{\partial A}(\sigma_{n+1}, A_{n+1}), \quad (25.4)$$

$$f(\sigma_{n+1}, A_{n+1}) \leq 0, \quad \xi_{n+1} \geq 0, \quad f(\sigma_{n+1}, A_{n+1})\xi_{n+1} = 0. \quad (25.5)$$

u_{n+1} と ξ_{n+1} が与えられたら、以下の2つの写像を定義します

$$\begin{aligned} \tilde{\mathcal{E}}^p &: (u_{n+1}, \theta \Delta t \xi_{n+1}, \zeta_n, \eta_n) \mapsto \varepsilon_{n+1}^p \\ \tilde{\mathcal{A}} &: (u_{n+1}, \theta \Delta t \xi_{n+1}, \zeta_n, \eta_n) \mapsto \alpha_{n+1} \end{aligned}$$

ここでペア $(\varepsilon_{n+1}^p, \alpha_{n+1}) = (\tilde{\mathcal{E}}^p(u_{n+1}, \theta \Delta t \xi_{n+1}, \zeta_n, \eta_n), \tilde{\mathcal{A}}(u_{n+1}, \theta \Delta t \xi_{n+1}, \zeta_n, \eta_n))$ は方程式 (25.3), (25.4) の解です (25.5) は考慮しません。)。後ほど、少なくとも単純な等方性の塑性流れの則では、これらの写像は単純な表現を持ち、場合によっては u_{n+1} に対し線形になることを示しています。

その場合でも u_{n+1} と ξ_{n+1} は応力 σ_{n+1} を与えます。

$$\sigma_{n+1} = \frac{\partial \psi^e}{\partial \varepsilon^e}(\varepsilon(u_{n+1}) - \varepsilon_{n+1}^p).$$

$$A_{n+1} = \frac{\partial \psi^p}{\partial \alpha}(\alpha_{n+1}).$$

相補性方程式 (25.5) は、[HA-WO2009] のように、 $r > 0$ に対し、適切に選択された相補性関数を使用して規定されます。

$$\int_{\Omega} (\xi_{n+1} - (\xi_{n+1} + r f(\sigma_{n+1}, A_{n+1}))_+) \lambda dx = 0, \forall \lambda$$

または

$$\int_{\Omega} (f(\sigma_{n+1} + (-f(\sigma_{n+1}, A_{n+1}) - \xi_{n+1}/r)_+, A_{n+1})) \lambda dx = 0, \forall \lambda$$

注記：表記法 $\Delta \xi_{n+1} = \Delta t \xi_{n+1}$ は書体でよく使われます。ここでの選択は、2つの量の間の差異を保存することです。これは主に、適応可能な時間ステップの使用のためです：時間ステップが変化するとき、値 ξ_n は、 θ -法の場合、 $\Delta \xi_n$ の代わりに ξ_n を使用することが望ましいです。

25.2.1 平面ひずみ近似

平面ひずみ近似は、長手方向の歪み（ z 軸に沿っていると仮定します）が他の方向の歪みに比べて小さく無視できると考えられる長い円筒形の物体の変形に対応する2次元問題です。これは、平面ひずみテンソルの形をとります

$$\varepsilon(u) = \begin{pmatrix} \varepsilon_{1,1} & \varepsilon_{1,2} & 0 \\ \varepsilon_{1,2} & \varepsilon_{2,2} & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

次の式を示します。

$$\bar{\varepsilon}(u) = \begin{pmatrix} \varepsilon_{1,1} & \varepsilon_{1,2} \\ \varepsilon_{1,2} & \varepsilon_{2,2} \end{pmatrix}$$

ひずみテンソルの無視されない成分を示します。ひずみテンソルの塑性と弾性部分の分解では、

$$\varepsilon_{1,3}^p = \varepsilon_{2,3}^p = \varepsilon_{1,3}^e = \varepsilon_{2,3}^e = 0$$

と次式となります。

$$\varepsilon_{3,3}^e + \varepsilon_{3,3}^p = \varepsilon_{3,3} = 0.$$

塑性モデルへの平面ひずみ近似の適応は、ほとんどの場合、簡単な作業です。等方性線形弾性応答は、

$$\sigma = \lambda \text{tr}(\varepsilon(u))I + 2\mu(\varepsilon(u) - \varepsilon^p),$$

であり、したがって

$$\bar{\sigma} = \lambda \text{tr}(\bar{\varepsilon}(u))\bar{I} + 2\mu(\bar{\varepsilon}(u) - \bar{\varepsilon}^p),$$

応力テンソルの非ゼロの $\sigma_{3,3}$ 成分は

$$\sigma_{3,3} = \lambda \text{tr}(\bar{\varepsilon}(u)) - 2\mu \varepsilon_{3,3}^p$$

均等塑性ひずみが想定される汎用的なケースでは、以下の通りになることに留意してください。

$$\text{tr}(\varepsilon^p) = 0 \quad \Rightarrow \quad \varepsilon_{3,3}^p = -(\varepsilon_{1,1}^p + \varepsilon_{2,2}^p).$$

25.2.2 平面応力近似

平面応力近似は、一般に、薄板の 2D 膜変形を記述します。この近似は、面内非ゼロ成分のみを有するように応力テンソルを規定します。

$$\sigma = \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & 0 \\ \sigma_{1,2} & \sigma_{2,2} & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

さらに、

$$\bar{\sigma} = \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} \\ \sigma_{1,2} & \sigma_{2,2} \end{pmatrix}$$

のように応力テンソルの面内成分を与えます。弾塑性の場合、一般に、付加的な未知変数 $\varepsilon_{1,3}^e, \varepsilon_{2,3}^e, \varepsilon_{3,3}^e$ により、応力テンソルの面外成分をゼロにするように 2 次の塑性流れ則を適用することにより成り立ちます。(例えば、[SO-PE-OW2008] を参照)。

等方性の線形弾性応答 $\sigma = \lambda \text{tr}(\varepsilon^e) + 2\mu \varepsilon^e$ の場合、次のようになります。

$$\varepsilon^e = \begin{pmatrix} \varepsilon_{1,1}^e & \varepsilon_{1,2}^e & 0 \\ \varepsilon_{1,2}^e & \varepsilon_{2,2}^e & 0 \\ 0 & 0 & \varepsilon_{3,3}^e \end{pmatrix}.$$

また

$$\varepsilon_{3,3}^e = -\frac{\lambda}{\lambda + 2\mu}(\varepsilon_{1,1}^e + \varepsilon_{2,2}^e)$$

そのため、

$$\bar{\sigma} = \lambda^* \text{tr}(\bar{\varepsilon}^e) + 2\mu \bar{\varepsilon}^e \quad \text{with } \lambda^* = \frac{2\mu\lambda}{\lambda + 2\mu} \quad (25.6)$$

さらに、

$$\|\text{Dev}(\sigma)\| = \left(\|\bar{\sigma}\|^2 - \frac{1}{3}(\text{tr}(\bar{\sigma}))^2 \right)^{1/2}. \quad (25.7)$$

均等塑性ひずみが仮定されている場合、以下が成り立ちます。

$$\text{tr}(\varepsilon^p) = 0 \quad \Rightarrow \quad \varepsilon_{3,3}^p = -(\varepsilon_{1,1}^p + \varepsilon_{2,2}^p).$$

25.3 いくつかの古典的な法則

Tresca : $\rho(\sigma) \leq \sigma_y$ 。ここで、 $\rho(\sigma)$ は Cauchy 応力テンソルのスペクトル半径、 σ_y は一軸降伏応力です。(これはいくつかの硬化内部変数に依存する可能性があります)。

Von Mises : $\|\text{Dev}(\sigma)\| \leq \sqrt{\frac{2}{3}}\sigma_y$ 。ここで $\text{Dev}(\sigma) = \sigma - \frac{1}{3}\text{tr}(\sigma)I$ は σ と $\|\sigma\| = \sqrt{\sigma:\sigma}$ の偏差部分です。

25.3.1 Von-Mises 基準 (Prandl-Reuss モデル) を伴う完全な等方性弾性熱可塑性

内部変数はなく、等方性弾性応答を考慮します。流動化則は

$$\dot{\varepsilon}^p = \gamma \frac{\text{Dev}(\sigma)}{\|\text{Dev}(\sigma)\|}$$

これは式 $\Psi(\sigma) = f(\sigma) = \|\text{Dev}(\sigma)\| - \sqrt{\frac{2}{3}}\sigma_y$ に対応します。

塑性流れ則の積分のための θ -法は次式となります。

$$\varepsilon_{n+1}^p - \varepsilon_n^p = (1 - \theta)\alpha(\sigma_n)\Delta t\xi_n \frac{\text{Dev}(\sigma_n)}{\|\text{Dev}(\sigma_n)\|} + \theta\alpha(\sigma_{n+1})\Delta t\xi_{n+1} \frac{\text{Dev}(\sigma_{n+1})}{\|\text{Dev}(\sigma_{n+1})\|}.$$

因子を選択すると、次のようになります。 $\alpha(\sigma_n) = \|\text{Dev}(\sigma_n)\|$ と $\xi_n = \frac{\gamma_n}{\alpha(\sigma_n)}$ 。ゆえに次式となります。

$$\varepsilon_{n+1}^p - \varepsilon_n^p = (1 - \theta)\Delta t\xi_n \text{Dev}(\sigma_n) + \theta\Delta t\xi_{n+1} \text{Dev}(\sigma_{n+1}).$$

$\text{Dev}(\sigma_{n+1}) = 2\mu\text{Dev}(\varepsilon(u_{n+1})) - 2\mu\varepsilon_{n+1}^p$ であるため次式となります。

$$\tilde{\varepsilon}^p(u_{n+1}, \theta\Delta t\xi_{n+1}, \zeta_n) = \zeta_n + \left(1 - \frac{1}{1 + 2\mu\theta\Delta t\xi_{n+1}}\right) (\text{Dev}(\varepsilon(u_{n+1})) - \zeta_n),$$

これは、 u_{n+1} (ただし、 ξ_{n+1} とは関係ありません) に関する線形表現です。

さらに、 ζ_n は次のように定義されます。

$$\zeta_n = \varepsilon_n^p + (1 - \theta)\Delta t\xi_n (\text{Dev}(\sigma_n)) = \varepsilon_n^p + (1 - \theta)\Delta t\xi_n 2\mu (\text{Dev}(\varepsilon(u_n)) - \varepsilon_n^p).$$

乗数の除去 (Return Mapping 法の場合)

次式が成り立ちます

$$\|\text{Dev}(\sigma_{n+1})\| = 2\mu\|\text{Dev}(\varepsilon(u_{n+1})) - \varepsilon_{n+1}^p\| = \frac{2\mu}{1 + 2\mu\theta\Delta t\xi_{n+1}}\|\text{Dev}(\varepsilon(u_{n+1})) - \zeta_n\|,$$

したがって、 $B = \text{Dev}(\varepsilon(u_{n+1})) - \zeta_n$

$$2\mu\|B\| \leq \sqrt{\frac{2}{3}}\sigma_y,$$

そして弾性範囲の場合は、 $\xi_{n+1} = 0$ 、または $\|\text{Dev}(\sigma_{n+1})\| = \sqrt{\frac{2}{3}}$ となり、

$$1 + 2\mu\theta\Delta t\xi_{n+1} = \frac{2\mu\|B\|}{\sqrt{\frac{2}{3}}\sigma_y},$$

であり、したがって

$$\varepsilon_{n+1}^p = \zeta_n + \left(1 - \sqrt{\frac{2}{3}} \frac{\sigma_y}{2\mu\|B\|}\right) B.$$

2つの選択肢は次のように表されます。

$$\varepsilon_{n+1}^p = \mathcal{E}^p(u_{n+1}, \zeta_n) = \zeta_n + \left(1 - \sqrt{\frac{2}{3}} \frac{\sigma_y}{2\mu\|B\|}\right)_+ B.$$

乗算器 ξ_{n+1} ($\theta \neq 1$ の θ -法である必要があります) は次のように与えられます。

$$\xi_{n+1} = \frac{1}{\theta\Delta t} \left(\sqrt{\frac{3}{2}} \frac{\|B\|}{\sigma_y} - \frac{1}{2\mu} \right)_+.$$

平面ひずみ近似

平面ひずみ近似 $\bar{\varepsilon}^p$ と $\bar{\varepsilon}(u_{n+1})$ は3次元歪みテンソルを面内歪みで置き換えるのと同じ表現をしています。

$$\bar{\varepsilon}^p(u_{n+1}, \theta\Delta t\xi_{n+1}, \bar{\zeta}_n) = \bar{\zeta}_n + \left(1 - \frac{1}{1 + 2\mu\theta\Delta t\xi_{n+1}}\right) (\overline{\text{Dev}}(\bar{\varepsilon}(u_{n+1})) - \bar{\zeta}_n),$$

ここで、 $\overline{\text{Dev}}(\bar{\varepsilon}) = \bar{\varepsilon} - \frac{\text{tr}(\bar{\varepsilon})}{3}\bar{I}$ は3D 偏差器の2D 制限です。

さらに、降伏条件については、

$$\|\text{Dev}(\sigma)\|^2 = 4\mu^2 \left(\|\overline{\text{Dev}}\bar{\varepsilon}(u) - \bar{\varepsilon}^p\|^2 + \left(\frac{\text{tr}(\bar{\varepsilon}(u))}{3} - \text{tr}(\bar{\varepsilon}^p) \right)^2 \right).$$

そして乗数の除去のために次式とします。

$$\bar{\varepsilon}^p(\bar{u}_{n+1}, \bar{\varepsilon}_n^p) = \bar{\zeta}_n^p + \left(1 - \sqrt{\frac{2}{3}} \frac{\sigma_y}{2\mu \|B\|}\right) \bar{B}$$

ここで $\bar{B} = \overline{\text{Dev}}(\bar{\varepsilon}(u_{n+1})) - \bar{\zeta}_n$ and $\|B\|^2 = \|\overline{\text{Dev}}(\bar{\varepsilon}(u_{n+1})) - \bar{\zeta}_n\|^2 + \left(\frac{\text{tr}(\bar{\varepsilon}(u_{n+1}))}{3} - \text{tr}(\bar{\zeta}_n)\right)^2$ 。

面応力近似

平面応力近似の場合、(25.6) を使用して、3D の場合の式から導きます。

$$\bar{\varepsilon}_{n+1}^p = \frac{1}{1 + 2\mu\theta\Delta\xi} \left(\bar{\zeta}_n + 2\mu\theta\Delta\xi \left(\bar{\varepsilon}(u_{n+1}) - \frac{2\mu}{3(\lambda + 2\mu)} (\text{tr}(\bar{\varepsilon}(u_{n+1})) - \text{tr}(\bar{\varepsilon}_{n+1}^p)) \bar{I} \right) \right),$$

ここで $\text{Dev}(\varepsilon(u)) = \varepsilon(u) - \frac{2\mu}{3(\lambda + 2\mu)} (\text{tr}(\varepsilon(u)) - \text{tr}(\bar{\varepsilon}^p))$ 。もちろん、この関係はまだ反転させなければなりません。 $\alpha = 1 + 2\mu\theta\Delta\xi$, $\beta = \frac{4\mu^2\theta\Delta\xi}{3\lambda + 6\mu}$ と $C = \bar{\zeta}_n + 2\mu\theta\Delta\xi \left(\bar{\varepsilon}(u_{n+1}) - \frac{2\mu}{3(\lambda + 2\mu)} (\text{tr}(\bar{\varepsilon}(u_{n+1}))) \bar{I} \right)$ を使用して表記します。

$$\bar{\varepsilon}_{n+1}^p = \frac{\beta \text{tr}(C)}{\alpha(\alpha - 2\beta)} \bar{I} + \frac{1}{\alpha} C.$$

さらに、降伏条件の場合、式 (25.7) を使用できます。

25.3.2 線形等方性および移動硬化と Von-Mises 基準を用いた等方性弾性率

等方性弾性応答と内部変数 $\alpha : \Omega \rightarrow \mathbb{R}$ を蓄積されたひずみとして次式を満たすと考えてみます。

$$\dot{\alpha} = \sqrt{\frac{2}{3}} \gamma$$

線形硬化は等方的硬化係数 H_i であり、次式となります。

$$\psi^p(\alpha) = \frac{1}{2} H_i \alpha^2$$

すなわち $A = H_i \alpha$ および一軸降伏応力は次のように定義されます。

$$\sigma_y(a) = \sigma_{y0} + A = \sigma_{y0} + H_i \alpha,$$

σ_{y0} は最初の一軸降伏応力です。降伏関数（およびこれは塑性モデル関連しているため、塑性ポテンシャル）は次のように定義されます。

$$\Psi(\sigma, A) = f(\sigma, A) = \|\text{Dev}(\sigma - \frac{2}{3}H_k\varepsilon^p)\| - \sqrt{\frac{2}{3}}(\sigma_{y0} + A),$$

ここで、 H_k は動力学的硬化係数です。前節と同じ計算により

$$\tilde{\mathcal{E}}^p(u_{n+1}, \theta\Delta t\xi_{n+1}, \zeta_n) = \zeta_n + \frac{1}{2(\mu + H_k/3)} \left(1 - \frac{1}{1 + 2(\mu + H_k/3)\theta\Delta t\xi_{n+1}}\right) (2\mu\text{Dev}(\varepsilon(u_{n+1})) - 2(\mu + H_k/3)\zeta_n)$$

$$\begin{aligned} \tilde{\mathcal{A}}(u_{n+1}, \theta\Delta t\xi_{n+1}, \zeta_n, \eta_n) &= \eta_n + \sqrt{\frac{2}{3}}\theta\Delta t\xi_{n+1}\|\text{Dev}(\sigma_{n+1} - \frac{2}{3}H_k\varepsilon_{n+1}^p)\| \\ &= \eta_n + \sqrt{\frac{2}{3}}\theta\Delta t\xi_{n+1}\|2\mu\text{Dev}(\varepsilon(u_{n+1})) - 2(\mu + H_k/3)\varepsilon_{n+1}^p\| \\ &= \eta_n + \sqrt{\frac{2}{3}}\frac{\theta\Delta t\xi_{n+1}}{1 + 2(\mu + H_k/3)\theta\Delta t\xi_{n+1}}\|2\mu\text{Dev}(\varepsilon(u_{n+1})) - 2(\mu + H_k/3)\zeta_n\| \\ &= \eta_n + \sqrt{\frac{2}{3}}\frac{1}{2(\mu + H_k/3)} \left(1 - \frac{1}{1 + 2(\mu + H_k/3)\theta\Delta t\xi_{n+1}}\right) \|2\mu\text{Dev}(\varepsilon(u_{n+1})) - 2(\mu + H_k/3)\zeta_n\| \end{aligned}$$

ここで、 ζ_n と η_n は次のように定義されます。

$$\zeta_n = \varepsilon_n^p + (1 - \theta)\Delta t\xi_n(\text{Dev}(\sigma_n) - \frac{2}{3}H_k\varepsilon_n^p) = \varepsilon_n^p + (1 - \theta)\Delta t\xi_n (2\mu\text{Dev}(\varepsilon(u_n)) - 2(\mu + H_k/3)\varepsilon_n^p),$$

$$\eta_n = \alpha_n + (1 - \theta)\sqrt{\frac{2}{3}}\Delta t\xi_n\|\text{Dev}(\sigma_n) - \frac{2}{3}H_k\varepsilon_n^p\| = \alpha_n + (1 - \theta)\sqrt{\frac{2}{3}}\Delta t\xi_n\|2\mu\text{Dev}(\varepsilon(u_n)) - 2(\mu + H_k/3)\varepsilon_n^p\|.$$

等方性硬化係数は、 $f(\sigma, A)$ 内のみで $\tilde{\mathcal{E}}^p(u_{n+1}, \theta\Delta t\xi, \varepsilon_n^p)$ の式には影響しないことに注意してください。

乗数の除去 (Return Mapping 法の場合)

式 $\delta = \frac{1}{1 + 2(\mu + H_k/3)\theta\Delta t\xi_{n+1}}$, $\beta = \frac{1 - \delta}{2(\mu + H_k/3)}$ と、 $B = 2\mu\text{Dev}(\varepsilon(u_{n+1})) - 2(\mu + H_k/3)\zeta_n$ によって ε_{n+1}^p と α_{n+1} の式は次のようになります。

$$\varepsilon_{n+1}^p = \zeta_n + \beta B, \quad \alpha_{n+1} = \eta_n + \sqrt{\frac{2}{3}}\beta\|B\|, \quad (25.8)$$

そして、塑性の拘束は次の通りです。

$$\delta\|B\| \leq \sqrt{\frac{2}{3}}(\sigma_{y0} + H_i\alpha_{n+1}).$$

従って、弾性の場合、 $\xi_{n+1} = 0, \delta = 1$ であり

$$\|B\| \leq \sqrt{\frac{2}{3}}(\sigma_{y0} + H_i \eta_n),$$

塑性領域の場合 $\xi_{n+1} > 0, \delta < 1$ 、 $\delta\|B\| = \sqrt{\frac{2}{3}}(\sigma_{y0} + H_i \alpha_{n+1})$ と $(1 - \delta)$ を使用して方程式を解きます。

$$\|B\| - (1 - \delta)\|B\| = \sqrt{\frac{2}{3}} \left(\sigma_{y0} + H_i \eta_n + \sqrt{\frac{2}{3}} \frac{H_i}{2(\mu + H_k/3)} (1 - \delta)\|B\| \right),$$

それにより、

$$1 - \delta = \frac{2(\mu + H_k/3)}{\|B\|(2\mu + \frac{2}{3}(H_k + H_i))} \left(\|B\| - \sqrt{\frac{2}{3}}(\sigma_{y0} + H_i \eta_n) \right)$$

この2つのケースをまとめると、

$$\beta = \frac{1}{\|B\|(2\mu + \frac{2}{3}(H_k + H_i))} \left(\|B\| - \sqrt{\frac{2}{3}}(\sigma_{y0} + H_i \eta_n) \right)_+$$

(25.8) により $\mathcal{E}^p(u_{n+1}, \zeta_n, \eta_n)$ と $\mathcal{A}(u_{n+1}, \zeta_n, \eta_n)$ が与えられます。乗算器 ξ_{n+1} は次のように与えられます。

$$\xi_{n+1} = \frac{1}{(2(\mu + H_k/3))\theta\Delta t} \left(\frac{1}{\delta} - 1 \right) = \frac{1}{\theta\Delta t} \frac{\beta}{1 - 2(\mu + H_k/3)\beta}.$$

平面ひずみ近似

$\delta = \frac{1}{1 + 2(\mu + H_k/3)\theta\Delta t\xi_{n+1}}$, $\beta = \frac{1 - \delta}{2(\mu + H_k/3)}$, $B = 2\mu\text{Dev}(\varepsilon(u_{n+1})) - 2(\mu + H_k/3)\zeta_n$ と $\bar{B} = 2\mu\overline{\text{Dev}}(\bar{\varepsilon}(u_{n+1})) - 2(\mu + H_k/3)\bar{\zeta}_n$ により面内部では

$$\bar{\mathcal{E}}^p(u_{n+1}, \theta\Delta t\xi_{n+1}, \bar{\zeta}_n) = \bar{\zeta}_n + \beta\bar{B},$$

$$\tilde{\mathcal{A}}(u_{n+1}, \theta\Delta t\xi_{n+1}, \zeta_n, \eta_n) = \eta_n + \sqrt{\frac{2}{3}}\beta\|B\|,$$

また

$$\|B\|^2 = \|2\mu\overline{\text{Dev}}(\bar{\varepsilon}(u_{n+1})) - 2(\mu + H_k/3)\bar{\zeta}_n\|^2 + \left(2\mu \frac{\text{tr}(\bar{\varepsilon}(u_{n+1}))}{3} - 2(\mu + H_k/3)\text{tr}(\bar{\zeta}_n) \right)^2.$$

降伏条件は

$$\delta \|B\| \leq \sqrt{\frac{2}{3}} (\sigma_{y0} + H_i \alpha_{n+1}).$$

β は、前の節で述べた式と同じ式 $\|B\|$ を持ちます。 $\bar{\zeta}_n$ と η_n の表現は、条件に応じて適用する必要があります。

25.3.3 Souza-Auricchio 弾塑性則（形状記憶合金用）

この流動化則の構成の正当性については、たとえば [GR-ST2015] を参照してください。 Von-Mises 応力基準と等方性弾性応答、内部変数および特殊タイプの移動硬化は、次のような制約を考慮して検討されます $\|\varepsilon^p\| \leq c_3$ 。塑性ポテンシャルと降伏関数は、

$$\Psi(\sigma) = f(\sigma) = \left\| \text{Dev} \left(\sigma - c_1 \frac{\varepsilon^p}{\|\varepsilon^p\|} - c_2 \varepsilon^p - \delta \frac{\varepsilon^p}{\|\varepsilon^p\|} \right) \right\| - \sqrt{\frac{2}{3}} \sigma_y,$$

相補性条件で

$$\delta \geq 0, \|\varepsilon^p\| \leq c_3, \delta(\|\varepsilon^p\| - c_3) = 0,$$

ここで、 c_1, c_2 と c_3 はいくつかの物理的なパラメータです。 $\frac{\varepsilon^p}{\|\varepsilon^p\|}$ は $\varepsilon^p = 0$ の単位球全体として処理されることに注意してください。

準備中 ...

25.4 弾塑性ブリック

テストプログラム `tests/plasticity.cc`, `interface/tests/matlab/demo_plasticity.m`, `interface/tests/matlab/demo_plasticity.py` と `contrib/test_plasticity` 内を参照してください。

25.4.1 汎用的なブリック

汎用的なブリックには 2 つのバージョンがあります。1 つめは塑性乗数が、問題の変数として保持されているときのもので、追加される項は次の形式です。

$$\int_{\Omega} \sigma_{n+1} : \nabla \delta u dx + \int_{\Omega} (\xi_{n+1} - (\xi_{n+1} + r f(\sigma_{n+1}, A_{n+1}))_+) \delta \xi dx = 0,$$

$r > 0$ がブリックによって (弾性係数の点で) 選択された特定の値を有し、Return Mapping 法が選択されたとき (塑性乗数は単なるデータです) は、次の項を追加するだけです。

$$\int_{\Omega} \sigma_{n+1} : \nabla v dx.$$

ブリックをモデル *md* に追加する関数は次の通りです。

```
getfem::add_small_strain_elastoplasticity_brick
  (md, mim, lawname, unknowns_type,
   const std::vector<std::string> &varnames,
   const std::vector<std::string> &params, region = size_type(-1));
```

ここで *lawname* は実装された塑性法則の名前であり、*unknowns_type* は塑性乗数が問題もしくは次の反復のために保存されたモデルのデータが (写像手法を返す) 未知数である離散化の選択を示します。いずれにしても乗数が格納されることに注意してください。 *varnames* は、塑性法則 (少なくとも変位、塑性乗数、塑性ひずみ) に依存する長さの変数とデータ名の集合です。 *params* はパラメータ (少なくとも弾性係数と降伏応力) 表現のリストです。これらの表現は、モデルのいくつかのデータ名 (または変数名) でもかまいませんが、弱形式言語の有効なスカラー式 (“1/2”, “2+sin(X[0])”, “1+Norm(v)” ... など) も可能です。 *params* にオプションで指定される最後の 2 つのパラメータは、塑性ひずみ積分と時間ステップ *dt* に使用される *theta* -法 (汎用された台形ルール) の *theta* パラメータです。省略された場合の *theta* のデフォルト値は 1 であり、これは 1 次整合性がある古典的な Backward Euler 方式に対応します。 *theta=1/2* は、2 次整合である Crank-Nicolson 法 (台形則) に対応します。 1/2 と 1 の間の値は有効な値です。 *dt* のデフォルト値は ‘timestep’ であり、モデルで定義された時間ステップを単に示すだけです (*md.set_time_step(dt)* による)。あるいは、任意の式 (データ名、定数値など ...) でもかまいません。時間ステップは、1 つの反復から次の反復まで変更することができます。 *region* はメッシュ領域です。

利用可能な塑性法則は次の通りです。

- “Prandtl Reuss” (または “等方性完全可塑性”)。硬化なしの等方性弾塑性。変数は変位、塑性乗数、塑性ひずみです。変位は変数でなければならず、前の時間ステップでの変位 (通常は “u” と “Previous_u”) に対応する “Previous_” の接頭辞に対応するデータがあります。塑性乗数には 2 つのバージョン (通常は “xi” と “Previous_xi”) が必要です。 *unknowns_type = DISPLACEMENT_ONLY* の場合はデータとして定義され、 *unknowns_type = DISPLACEMENT_AND_PLASTIC_MULTIPLIER* の場合は変数として定義されます。塑性ひずみは、*mesh_fem* または (好ましくは) *im_data* (*mim* に対応する) に格納された $n \times n$ のデータテンソルフィールドを表す必要があります。データは、第 1 のパラメータ (Lame 係数)、第 2 のパラメータ (せん断弾性率) および一軸降伏応力です。重要: この規則は 3D 表現を実装していることに注意してください。 2D で使用されている場合、式は単に 2D に転置されます。平面ひずみ近似については、以下を参照してください。
- “平面ひずみ Prandtl Reuss” (または “平面ひずみ等方性完全塑性”) 平面ひずみ近似に適合したものと同じ規則。 2D でのみ使用できます。
- “Prandtl Reuss 線形硬化” (または “等方性塑性線形硬化”)。線形等方性および移動硬化による等方性弾性

塑性。“Prandtl Reuss” 則と比較した追加変数：累積塑性ひずみ。塑性ひずみと同様に、時間ステップの終了時にのみ記憶されるので、単純なデータが必要である（`im_data` 上にあるのが好ましい）。2つの追加パラメータ：移動硬化係数および等方性パラメータ。3D 表現のみです。

- ”平面ひずみ Prandtl Reuss 線形硬化”（または“平面ひずみ等方性可塑性線形硬化”）。前のものと同じ規則ですが、平面ひずみ近似に適合しています。2D でのみ使用できます。

重要：`small_strain_elastoplasticity_next_iter` は、各時間ステップの終わり、次のステップの前（および後処理の前に、これが塑性ひずみおよび塑性乗数の値を設定する）で呼び出されなければならないことを忘れないでください。

さらに、以下の関数は、微小塑性ひずみブリックの時間ステップをすすめることを許可します：

```
getfem::small_strain_elastoplasticity_next_iter
(md, mim, lawname, unknowns_type,
  const std::vector<std::string> &varnames,
  const std::vector<std::string> &params, region = size_type(-1));
```

パラメータは `add_small_strain_elastoplasticity_brick` のパラメータとまったく同じでなければなりません。そのため、この関数の説明を参照してください。基本的には、このブリックは塑性ひずみと塑性積演算子を計算し、次のステップのために保存します。さらに、計算された変位を前の時間ステップの変位を格納するデータに（通常は“`u`” から“`Previous_u`” に）コピーします。この関数は `compute_small_strain_elastoplasticity_Von_Mises` を使う前に呼び出さなければなりません。

次の関数は

```
getfem::compute_small_strain_elastoplasticity_Von_Mises
(md, mim, lawname, unknowns_type,
  const std::vector<std::string> &varnames,
  const std::vector<std::string> &params,
  const mesh_fem &mf_vm, model_real_plain_vector &VM,
  region = size_type(-1));
```

`mf_vm` で近似された微小ひずみ弾塑性項に対する Von Mises 応力場を計算し、その結果を“`VM`”に格納します。他のすべてのパラメータは `add_small_strain_elastoplasticity_brick` とまったく同じです。`small_strain_elastoplasticity_next_iter` は、この関数を呼び出す前に呼び出さなければならないことに注意してください。

25.4.2 完全な可塑性のための低レベル汎用構築に基づく特定のブリック

これは、等方性完全塑性に制限され、低レベルの汎用的な構築に基づいている弾塑性要素の以前のバージョンです。流れ則が有限要素節点（Gauss 点ではない）に積分されていることはテストにおいて注意する点です。

このブリックをモデルに追加する関数は以下の通りです。

```
getfem::add_elastoplasticity_brick
    (md, mim, ACP, varname, previous_varname, datalambda, datamu, datathreshold, datasigma, region);
```

ここで

- `varname` はブリックが追加されたメインの未知変位 (u) を表します。
- `previous_varname` は直前の時間ステップでの変位です。
- `datalambda` と `datamu` は **Lame** 係数に対応するデータです。
- `datathreshold` は、調査した材料の塑性閾値を表します。
- `datasigma` は、材料によって支持される応力制約値を表します。使用される **Newton** 法に必要な時間スキームの 2 つの反復で構成される必要があります。 `datasigma` が定義されている有限要素法は `varname` の導関数を表すことができることに注意してください。
- `ACP` は、使用される投影のタイプに対応します。これは `abstract_constraints_projection` 型を持ち、現時点では、**Von Mises** のものに対応する `VM_projection` しか存在しません。

注意: `datalambda`, `datamu` と `datathreshold` は定数でも、同じ有限要素法で記述してもかまいません。

この関数は、**Newton** 法を使用して解かれる接ベクトルと右辺ベクトルを構築します。

さらに、次の関数は

```
getfem::elastoplasticity_next_iter
    (md, mim, varname, previous_varname, ACP, datalambda, datamu, datathreshold, datasigma);
```

載荷または除荷後に材料がサポートしている新しい応力制約値を計算し、(一度解を求めたら) 次のように変数 `varname` および `datasigma` をアップロードします。

$$u^{n+1} \Rightarrow u^n \quad \text{and} \quad \sigma^{n+1} \Rightarrow \sigma^n$$

そして、 u^n と σ^n は計算された新しい値を含み、プロセスを再起動することができます。

次の関数は

```
getfem::compute_elastoplasticity_Von_Mises_or_Tresca
    (md, datasigma, mf_vm, VM, tresca=false);
```

`datasigma` に格納されている **Von Mises** (または `tresca = true` の場合 **Tresca**) 基準の応力テンソルのを計算します。応力は `mesh_fem` `mf_vm` で評価され、ベクトル `VM` に格納されます。もちろん、この関数は以前の関数 `elastoplasticity_next_iter` がこの関数の前に呼び出された場合にのみ使用できます。

次の関数は

```
getfem::compute_plastic_part  
  (md, mim, mf_pl, varname, previous_varname, ACP, datalambda, datamu, datathreshold, datasigma, P
```

载荷と除荷後に出現する材料の可塑性部分を `Plast` に `mf_pl` で計算します。

`datasigma` は、新しい応力制約値を含むベクトル、すなわち、材料の载荷または除荷後のベクトルでなければならぬことに留意してください。

第 26 章

剛体運動が大きい物体の ALE サポート

26.1 物体を回転させるための ALE 項

このセクションでは、回転対称性（電車の車輪が典型）を有する回転体のための ALE 定式化の使用を容易にするブリックの一覧を紹介します。

26.1.1 理論的背景

この手法の戦略は、回転対称性を有する回転体のための Euler と Lagrangian の間の中間的な方法を採用することにあります。この中間的な方法は、参照構成に対して回転する軸で構成されています。たとえば、[Dr-La-Ek2014] と [Nackenhurst2004] を参照してください。

検討対象の物体は、ほぼ剛体運動であると考えます。

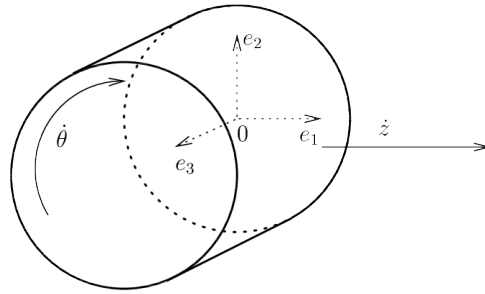
$$\tau(X) = R(t)X + Z(t)$$

この剛体運動に対して追加の（より小さいと予想される）変形を有することができます。ここで、 $R(t)$ は回転行列です

$$R(t) = \begin{pmatrix} \cos(\theta(t)) & \sin(\theta(t)) & 0 \\ -\sin(\theta(t)) & \cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

そして $Z(t)$ は変形です。転がり接触の正の角度に対する正の平行移動と一致するためには、回転は時計回りであることに留意してください。これを次の図に示します。

なお、説明は 3 次元体についての説明です。2 次元の物体の場合、第 3 軸は無視されます。 $R(t)$ は 2×2 回転行



列です。 $r(t)$ 回転が以下のような場合について考えます :

$$r(t, X) = R(t)X, \quad \text{and } A = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

すると

$$\dot{r}(t, X) = \dot{\theta} A R(t) X$$

もし $\varphi(t, X)$ が Ω^0 の参照構成から時刻 t の変形設定 Ω_t へマップしている物体の変形である場合、ALE 記述は、円筒の変形の分解に帰着されます。

$$\varphi(t, X) = (\tau(t) \circ \bar{\varphi}(t) \circ r(t))(X) = \bar{\varphi}(t, r(t, X)) + Z(t)$$

$\bar{X} = R(t)X$ とすると、新しく考える変形は

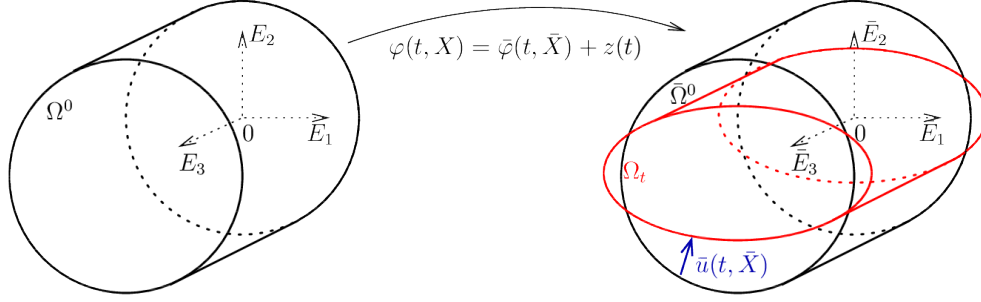
$$\bar{\varphi}(t, \bar{X}) = \varphi(X) - Z(t)$$

参照構成 Ω^0 の回転対称性により、 $\bar{\Omega}^0 = r(t, \Omega^0)$ は t が独立であり、新しい参照構成として機能します。これを次の図に示します。

この方法の ALE の名称は、次の事実によって正当化されます。 $\bar{\Omega}^0$ は、剛体運動のための Euler 型の間中構成であり、固体の追加の変形のための Lagrangian 型です。この場合、

$$\bar{u}(t, \bar{X}) = \bar{\varphi}(t, \bar{X}) - \bar{X}$$

剛性体の動きに対するこの追加の変位が小さい場合、微小変形モデル（例えば線形弾性）を使用することができるという利点があります。



標準的な手法のオブジェクト化は、これらの手法の中間構成は時間的に不変であるため、変更された時間の導関数を表現することを除いて、標準的な参照形態の表現とほとんど同じです。

$$\frac{\partial \varphi}{\partial t} = \frac{\partial \bar{\varphi}}{\partial t} + \dot{\theta} \nabla \bar{\varphi} A \bar{X} + \dot{Z}(t),$$

$$\frac{\partial^2 \varphi}{\partial t^2} = \frac{\partial^2 \bar{\varphi}}{\partial t^2} + 2\dot{\theta} \nabla \frac{\partial \bar{\varphi}}{\partial t} A \bar{X} + \dot{\theta}^2 \operatorname{div}((\nabla \bar{\varphi} A \bar{X}) \otimes (A \bar{X})) + \ddot{\theta} \nabla \bar{\varphi} A \bar{X} + \ddot{Z}(t).$$

項 $\dot{\theta} A \bar{X} = \begin{pmatrix} \dot{\theta} \bar{X}_2 \\ -\dot{\theta} \bar{X}_1 \\ 0 \end{pmatrix}$ が剛体運動速度ベクトルであることに注意してください。今、 $\Theta(t, X)$ が物質点（例えば温度）に接続された量であり、次に $\bar{\Theta}(t, \bar{X}) = \Theta(t, X)$ の場合、単純に次の式になります

$$\frac{\partial \Theta}{\partial t} = \frac{\partial \bar{\Theta}}{\partial t} + \dot{\theta} \nabla \bar{\Theta} A \bar{X}$$

これは、時間微分が考慮されたモデルに介入し進展する変数ごとに補正が行われなければならないことに注意してください（例えば、可塑性のために板状の流れを考える）。そのため、特定のモデルの要素を直接使用することはできません（たとえば、可塑ブリックなど）。

GetFEM++ 構造的力学モデルのためのブリックは、主に、未知変数として変位を考慮しています。変位の式は次のとおりです。

$$\frac{\partial u}{\partial t} = \frac{\partial \bar{u}}{\partial t} + \dot{\theta} (I_d + \nabla \bar{u}) A \bar{X} + \dot{Z}(t),$$

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 \bar{u}}{\partial t^2} + 2\dot{\theta} \nabla \frac{\partial \bar{u}}{\partial t} A \bar{X} + \dot{\theta}^2 \operatorname{div}(((I_d + \nabla \bar{u}) A \bar{X}) \otimes (A \bar{X})) + \ddot{\theta} (I_d + \nabla \bar{u}) A \bar{X} + \ddot{Z}(t).$$

過渡項の弱定式化

ρ^0 を、回転対称性を有する基準構成における密度と仮定すると、加速度に対応する弱定式項は ($v(X) = \bar{v}(\bar{X})$ は試行関数) :

$$\int_{\Omega^0} \rho^0 \frac{\partial^2 u}{\partial t^2} \cdot v dX = \int_{\Omega^0} \rho^0 \left[\frac{\partial^2 \bar{u}}{\partial t^2} + 2\dot{\theta} \nabla \frac{\partial \bar{u}}{\partial t} A\bar{X} + \dot{\theta}^2 \operatorname{div}(((I_d + \nabla \bar{u})A\bar{X}) \otimes (A\bar{X})) + \ddot{\theta}(I_d + \nabla \bar{u})A\bar{X} + \ddot{Z}(t) \right] \cdot \bar{v} d\bar{X}.$$

右辺の第3項は、以下のように部分積分することができます。

$$\begin{aligned} \int_{\Omega^0} \rho^0 \dot{\theta}^2 \operatorname{div}(((I_d + \nabla \bar{u})A\bar{X}) \otimes (A\bar{X})) \cdot \bar{v} d\bar{X} &= - \int_{\bar{\Omega}^0} (\dot{\theta}^2 (I_d + \nabla \bar{u})A\bar{X}) \cdot (\nabla(\rho^0 \bar{v}))A\bar{X} d\bar{X} \\ &\quad + \int_{\partial \bar{\Omega}^0} \rho^0 \dot{\theta}^2 (((I_d + \nabla \bar{u})A\bar{X}) \otimes (A\bar{X})) \bar{N} \cdot \bar{v} d\bar{\Gamma}. \end{aligned}$$

$\partial \bar{\Omega}^0$ 上の外向きの単位法線ベクトル \bar{N} は $A\bar{X}$ に直交し境界項はゼロであり $\nabla(\rho^0 \bar{v}) = \bar{v} \otimes \nabla \rho^0 + \rho^0 \nabla \bar{v}$ であり $\nabla \rho^0 \cdot (A\bar{X}) = 0$ であり ρ^0 は回転対称性を持つ仮定のため、

$$\int_{\Omega^0} \rho^0 \dot{\theta}^2 \operatorname{div}(((I_d + \nabla \bar{u})A\bar{X}) \otimes (A\bar{X})) \cdot \bar{v} d\bar{X} = - \int_{\bar{\Omega}^0} \rho^0 \dot{\theta}^2 (\nabla \bar{u} A\bar{X}) \cdot (\nabla \bar{v} A\bar{X}) d\bar{X} - \int_{\bar{\Omega}^0} \rho^0 \dot{\theta}^2 (A^2 \bar{X}) \cdot \bar{v} d\bar{X}.$$

したがって、全体では

$$\begin{aligned} \int_{\Omega^0} \rho^0 \frac{\partial^2 u}{\partial t^2} \cdot v dX &= \int_{\bar{\Omega}^0} \rho^0 \left[\frac{\partial^2 \bar{u}}{\partial t^2} + 2\dot{\theta} \nabla \frac{\partial \bar{u}}{\partial t} A\bar{X} + \ddot{\theta} \nabla \bar{u} A\bar{X} \right] \cdot \bar{v} d\bar{X} \\ &\quad - \int_{\bar{\Omega}^0} \rho^0 \dot{\theta}^2 (\nabla \bar{u} A\bar{X}) \cdot (\nabla \bar{v} A\bar{X}) d\bar{X} - \int_{\bar{\Omega}^0} \rho^0 (\dot{\theta}^2 A^2 \bar{X} + \ddot{\theta} A\bar{X} + \ddot{Z}(t)) \cdot \bar{v} d\bar{X}. \end{aligned}$$

2つの項が問題の保磁力を悪化させることがあり、したがってその適切な負担と時間積分スキームの安定性を低下させる可能性があることに注意してください：第2項(対流項)と第5項。これらの項は、角速度 $\dot{\theta}$ の大きな値に対して追加の安定化技法を使用することを必要とするかもしれません。

26.1.2 利用可能なブリック...

次のように使用します:

```
ind = getfem::brick_name(parameters);
```

ここで parameters はパラメータです...

26.2 一様変形物体の一部の ALE 項

このセクションでは、ある方向に無限であり、着目する部分（計算が考慮される）がその方向（通常はバー）で均一に変形する物体に対して ALE 式の使用を容易にするブリックの一覧を提示します。

26.2.1 理論的背景

参照構成 $\Omega^0 \in \mathbb{R}^d$ が E_1 の方向で無限境界の物体を考えてみます、 $\Omega^0 = \mathbb{R} \times \omega^0$ ただし、 $\omega^0 \in \mathbb{R}^{d-1}$ になります。時刻 t において、この物体の“窓”だけが考慮されます。

$$\Omega^{0t} = (\alpha + z(t), \beta + z(t)) \times \omega^0$$

ここで、 $z(t)$ は変形を表します。

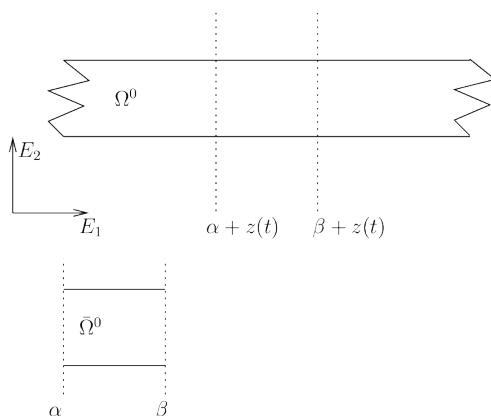
$\varphi(t, X)$ が時刻 t における参照構成 Ω^0 を変形された構成 Ω_t に写像する物体の変形である場合、ALE の記述は、中間参照構成を考慮したもので構成されます。

$$\bar{\Omega}^0 = (\alpha, \beta) \times \omega^0$$

そして $\bar{\varphi}(t, X) : \mathbb{R}_+ \times \bar{\Omega}^0 \rightarrow \mathbb{R}^d$ は次で定義されています

$$\bar{\varphi}(t, \bar{X}) = \varphi(t, X), \quad \text{with } \bar{X} = X - Z(t),$$

ここで、 $Z(t) = z(t)E_1$ 。 $\bar{\Omega}^0$ の着目点は時間に依存しないことです。いくつかの特別な境界条件（吸収周期的な境界条件）は、 $\{\alpha\} \times \omega^0$ と $\{\beta\} \times \omega^0$ 上で定義され、物体の無限であるという条件を近似します。



以下の式が示されます

$$\bar{u}(t, \bar{X}) = \bar{\varphi}(t, \bar{X}) - X = u(t, X),$$

中間構成上の変位を確認するのは簡単です

$$\begin{aligned}\frac{\partial \varphi}{\partial t} &= \frac{\partial \bar{u}}{\partial t} - \nabla \bar{u} \dot{Z} \\ \frac{\partial^2 \varphi}{\partial t^2} &= \frac{\partial^2 \bar{u}}{\partial t^2} - \nabla \frac{\partial \bar{u}}{\partial t} \dot{Z} + \frac{\partial^2 \bar{u}}{\partial \dot{Z}^2} - \nabla \bar{u} \ddot{Z}.\end{aligned}$$

過渡項の弱定式化

参照の ρ^0 の密度は、考慮された変換で不変であると仮定すると、弱定式化における加速度に対応する項の、試行関数 ($v(X) = \bar{v}(\bar{X})$) と部分積分済みのものは：



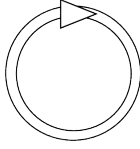
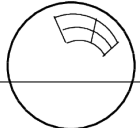

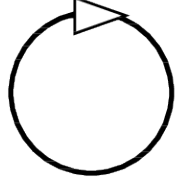

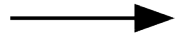


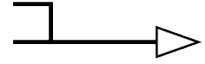

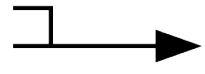
$$\begin{aligned}& \int_{\Omega^0} \rho^0 \frac{\partial^2 u}{\partial t^2} \cdot v dX = \\ & \int_{\bar{\Omega}^0} \rho^0 \left[\frac{\partial^2 \bar{u}}{\partial t^2} - 2\nabla \frac{\partial \bar{u}}{\partial t} \dot{Z} - \nabla \bar{u} \ddot{Z} \right] \cdot \bar{v} - \rho^0 (\nabla \bar{u} \dot{Z}) \cdot (\nabla \bar{v} \dot{Z}) d\bar{X} + \int_{\partial \bar{\Omega}^0} \rho^0 (\nabla \bar{u} \dot{Z}) \cdot \bar{v} (\dot{Z} \cdot \bar{N}) d\bar{\Gamma},\end{aligned}$$

ここで、 \bar{N} は $\partial \bar{\Omega}^0$ の外向きの単位法線ベクトルです。最後の項は $(\alpha, \beta) \times \partial \omega^0$ のように消えます。しかし $\{\alpha\} \times \omega^0$ と $\{\beta\} \times \omega^0$ 上では必要ありません。

第 27 章

付録 A. 有限要素法リスト

表 27.1 自由度の種類を表す記号

<p>●</p> <p>節点での関数の値。</p> <p></p> <p>3D 要素の第 3 座標に沿った勾配の値。</p> <p></p> <p>第 1 座標に沿った 2 次導関数の値 (2 回)。</p> <p></p> <p>節点での 2 次導関数 (hessian) 全体の値。</p> <p></p>	<p></p> <p>第 1 座標に沿った勾配の値。</p> <p></p> <p>節点での勾配全体の値。</p> <p></p> <p>第 2 座標に沿った 2 次導関数の値 (2 回)。</p> <p></p> <p>ベクトル要素のあるベクトル (たとえば、エッジ) を持つスカラー積。</p> <p></p>	<p></p> <p>第 2 座標に沿った勾配の値。</p> <p></p> <p>面への正規微分値。</p> <p></p> <p>3D で第 3 座標 (2 倍) に沿った 2D または 2 次導関数の 2 次交差導関数の値。</p> <p></p> <p>ベクトル要素の面に垂直なスカラー積。</p>
<p>208</p> <p>指定する要素または面の Bubble 関数。</p>	<p>詳細空間内の Lagrange ハイアラーキ自由度節点での値。</p>	<p>第 27 章 付録 A. 有限要素法リスト</p>

GetFEM++ で定義されているすべての有限要素法は、`getfem_fem.h` ファイルで宣言され、有限要素法記述子は次の関数に含まれます。

```
getfem::pfem pf = getfem::fem_descriptor("name of method");
```

ここで、"name of method" は既存のメソッドの中から選択される文字列です。

27.1 シンプレックスの古典的 Lagrange 要素 P_K

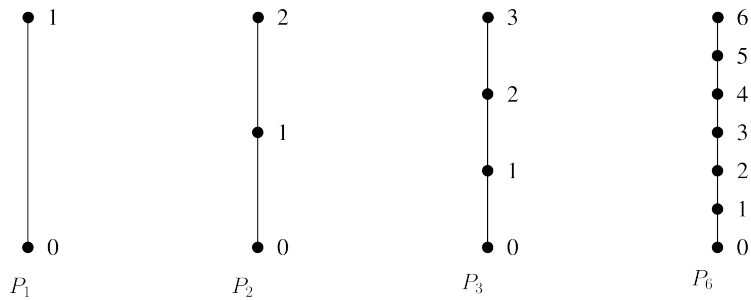
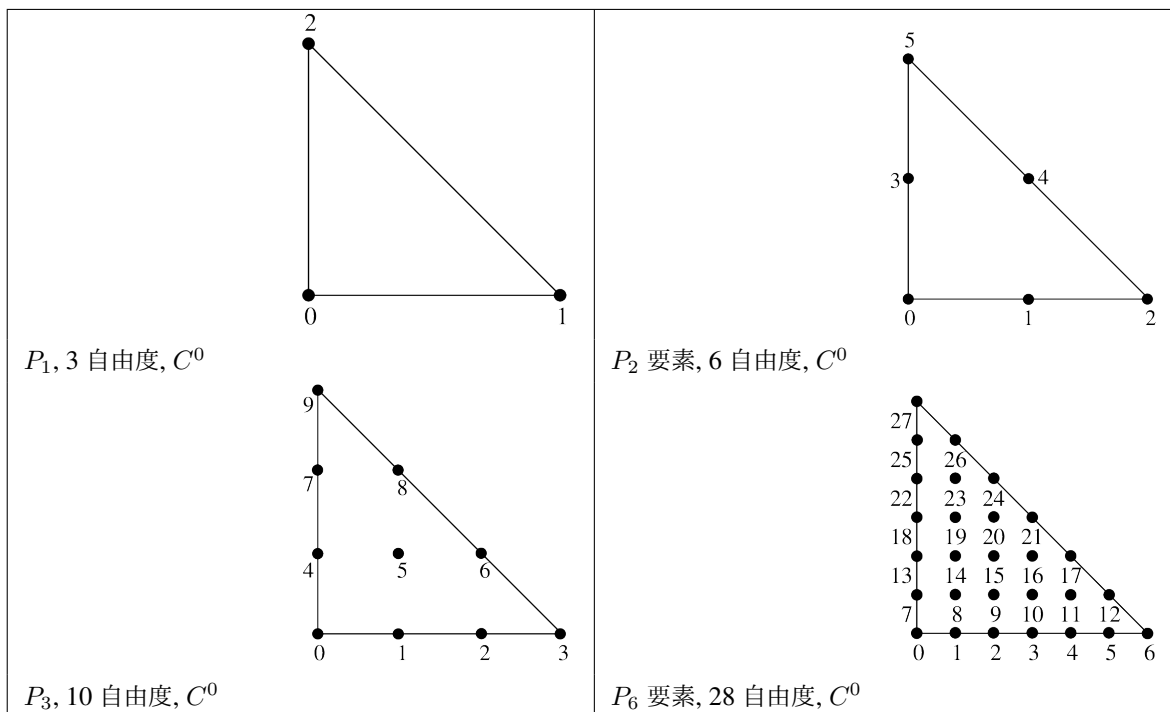


図 27.1 セグメント上の古典的な Lagrange 要素 P_K の例

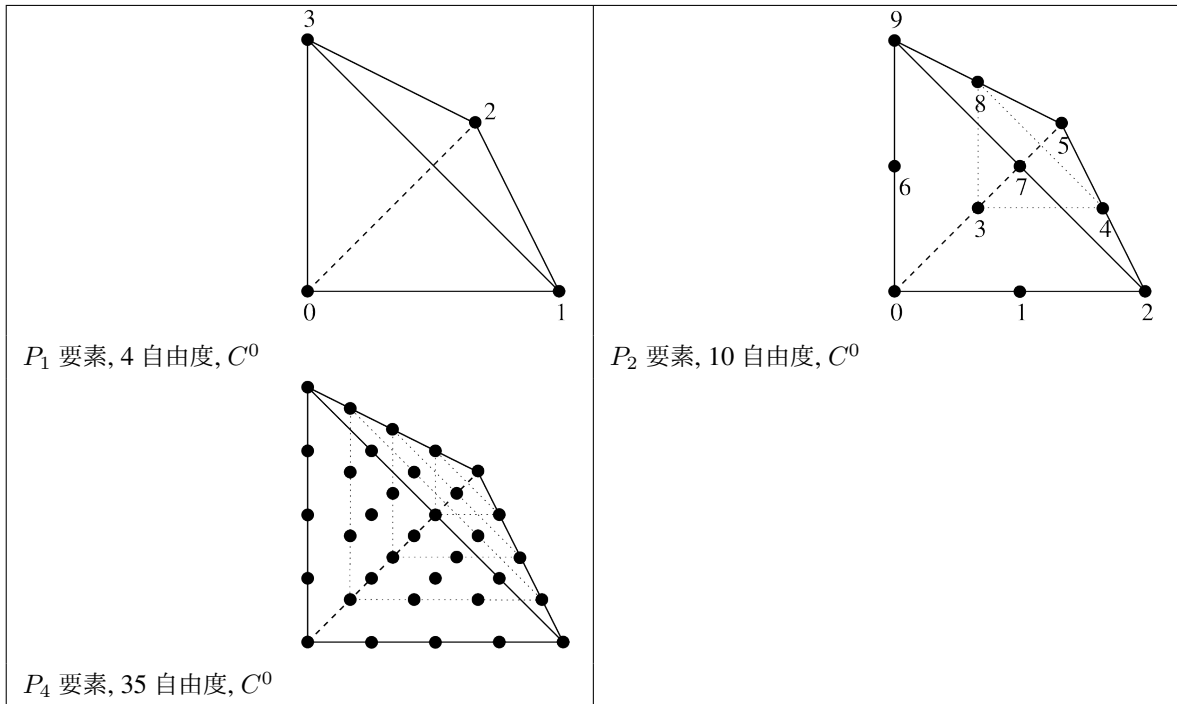
任意の次元と任意の次数の古典的な Lagrange 要素 P_K を定義することが可能です。このような要素の各自由度は、対応する節点上の関数の値に対応します。節点のグリッドは、いわゆる Lagrange グリッドです (図 *セグメント上の古典的な Lagrange 要素 の例*)。

表 27.2 3 角形上の古典的な Lagrange 要素 の例。



次元 P と次数 K の古典的な Lagrange 要素 P_K の自由度の数は $\frac{(P+K)!}{P!K!}$ です。たとえば、次元 2、($P=2$) の場合、この値は $\frac{(K+1)(K+2)}{2}$ で、次元 3 の場合 $\frac{(K+1)(K+2)(K+3)}{6}$ です。

表 27.3 4 面体上の古典的な Lagrange 要素 の例。



GetFEM++ で使われる節点を数値化するための具体的例として、図 セグメント、図 3 角形 と図 4 面体 があります。数値表現をすると、

$$i_0, i_1, \dots, i_P,$$

インデックスは

$$0 \leq i_0, i_1, \dots, i_P \leq K, \text{ and } \sum_{n=0}^P i_n = K.$$

節点の座標は次のように計算されます。

$$a_{i_0, i_1, \dots, i_P} = \sum_{n=0}^P \frac{i_n}{K} S_n, \text{ for } K \neq 0,$$

ここで、 S_0, S_1, \dots, S_N は、シンプレックスの頂点です ($K = 0$ の特定の場合 $a_{0,0,\dots,0} = \sum_{n=0}^P \frac{1}{P+1} S_n$ となります)。次に、各節点 a_{i_0, i_1, \dots, i_P} に対応する各基底関数は

$$\phi_{i_0, i_1, \dots, i_P} = \prod_{n=0}^P \prod_{j=0}^{i_n-1} \left(\frac{K\lambda_n - j}{j+1} \right).$$

ここで、 λ_n は重心座標です。つまり次のような次数 1 の多項式です。頂点 S_n での値は 1 そしてその値は他の頂点では 0 です。参照要素では、

$$\lambda_n = x_n, \quad 0 \leq n < P,$$

$$\lambda_P = 1 - x_0 - x_1 - \dots - x_{P-1}.$$

同じ角度の2つの要素の間に（異なる寸法であっても）、自由度要素はクラス C^0 です。これは、グローバル多項式が連続的であることを意味します。異なる度合いの要素をリンクしようとすると、リンクされていない自由度の問題が発生します。これは *GetFEM++* によって自動的にサポートされるわけではないので、サポートする必要があります（これらに自由度の制約を追加してください）。

いくつかの使用法（例えば *gradient* の計算）では、リンクされる共通の面の自由度がないかもしれません。そのため、古典的な P_K Lagrange 要素には2つのバージョンが存在します。

表 27.4 古典的 Lagrange 要素 "FEM_PK(P, K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
$K, 0 \leq K \leq 255$	$P, 1 \leq P \leq 255$	$\frac{(K+P)!}{K!P!}$	C^0	いいえ ($Q = 1$)	はい ($M = Id$)	はい

表 27.5 不連続 Lagrange 要素 "FEM_PK_DISCONTINUOUS(P, K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
$K, 0 \leq K \leq 255$	$P, 1 \leq P \leq 255$	$\frac{(K+P)!}{K!P!}$	不連続な	いいえ ($Q = 1$)	はい ($M = Id$)	はい

たとえ Lagrange 要素が任意の次数に対して定義されていても、Lagrange 基底の“ノイズの多い”特性のために、高い次数を選択することは、多数の適用例にとって問題となる可能性があります。これらの要素は、微分方程式の基本的な補間が推奨されており、ハイラーキ基底の要素の適用が望ましいです（対応する節を参照）。

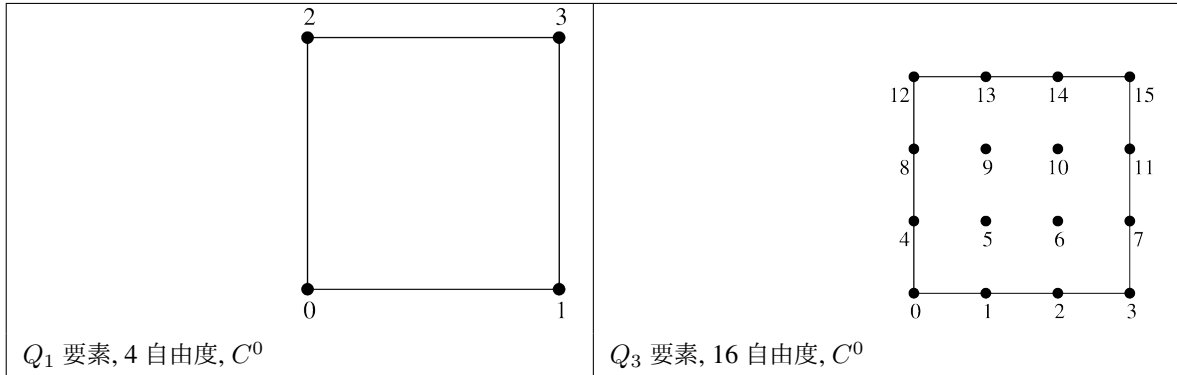
27.2 他のジオメトリ上の古典的な Lagrange 要素

平行6面体またはプリズムの古典的な Lagrange 要素は、シンプレックス上の Lagrange 要素のテンソル積として得られます。2つの要素が定義されている場合、次元 P^1 と次元 P^2 の、(参照要素上の) テンソル積の基底関数は

$$\hat{\varphi}_{ij}(x, y) = \hat{\varphi}_i^1(x) \hat{\varphi}_j^2(y), \quad x \in \mathbb{R}^{P^1}, y \in \mathbb{R}^{P^2},$$

ここで、 $\hat{\varphi}_i^1$ 、および $\hat{\varphi}_i^2$ は、それぞれ第1要素と第2要素の基本関数です。

表 27.6 古典的な次元 2 の Lagrange 要素の例。



P 次の次元の平行 6 面体上の Q_K 要素は、はセグメント上の古典的 P_K 要素のテンソル積 P として得られます。次数 2 の例は図 [次数 2](#) に、次数 3 は図 [次数 3](#) に示されています。

$P > 1$ 次のプリズムは、 $P - 1$ 次のシンプレックスとセグメントの直積です。このプリズム上の $P_K \otimes P_K$ 要素は、古典的な $P - 1$ 次の単項式上の古典的な P_K 要素とセグメント上の $P - 1$ 次の古典的な P_K 要素のシンプレックスのテンソル積です。 $P = 2$ の場合は平行 6 面体と一致します。次数 3 の例を図 [次数 3](#) に示します。これは、各次元で同じ次数を持たないことも可能です。例を図 [次数 3, プリズム](#) に示します。

表 27.7 次元 3 における古典的 Lagrange 要素の例。

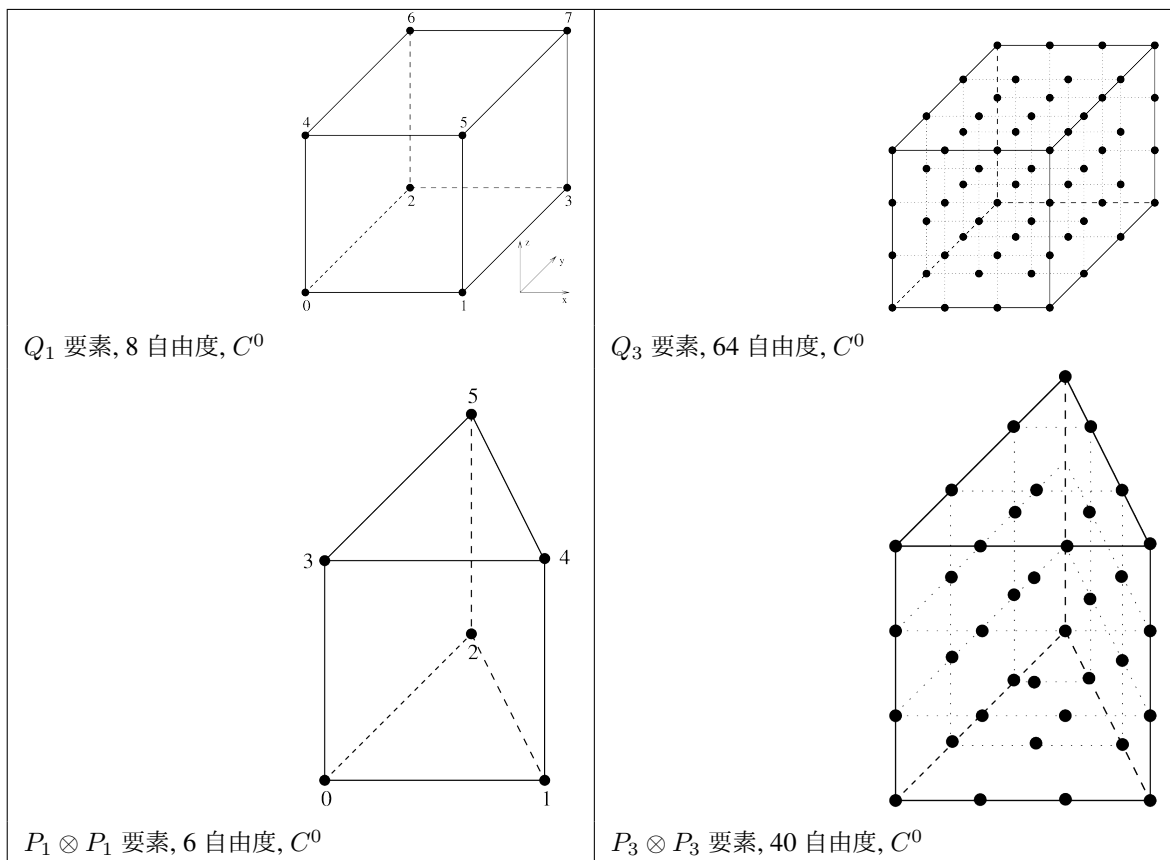


表 27.8 . 平行四辺形の Lagrange 要素 "FEM_QK(P, K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
$KP, 0 \leq K \leq 255$	$P, 1 \leq P \leq 255$	$(K + 1)^P$	C^0	いいえ ($Q = 1$)	はい ($M = Id$)	はい

表 27.9 . プリズム上の Lagrange 要素 "FEM_PK_PRISM(P, K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
$2K, 0 \leq K \leq 255$	$P, 2 \leq P \leq 255$	$(K + 1) \times \frac{(K + P - 1)!}{K!(P - 1)!}$	C^0	いいえ ($Q = 1$)	はい ($M = Id$)	はい

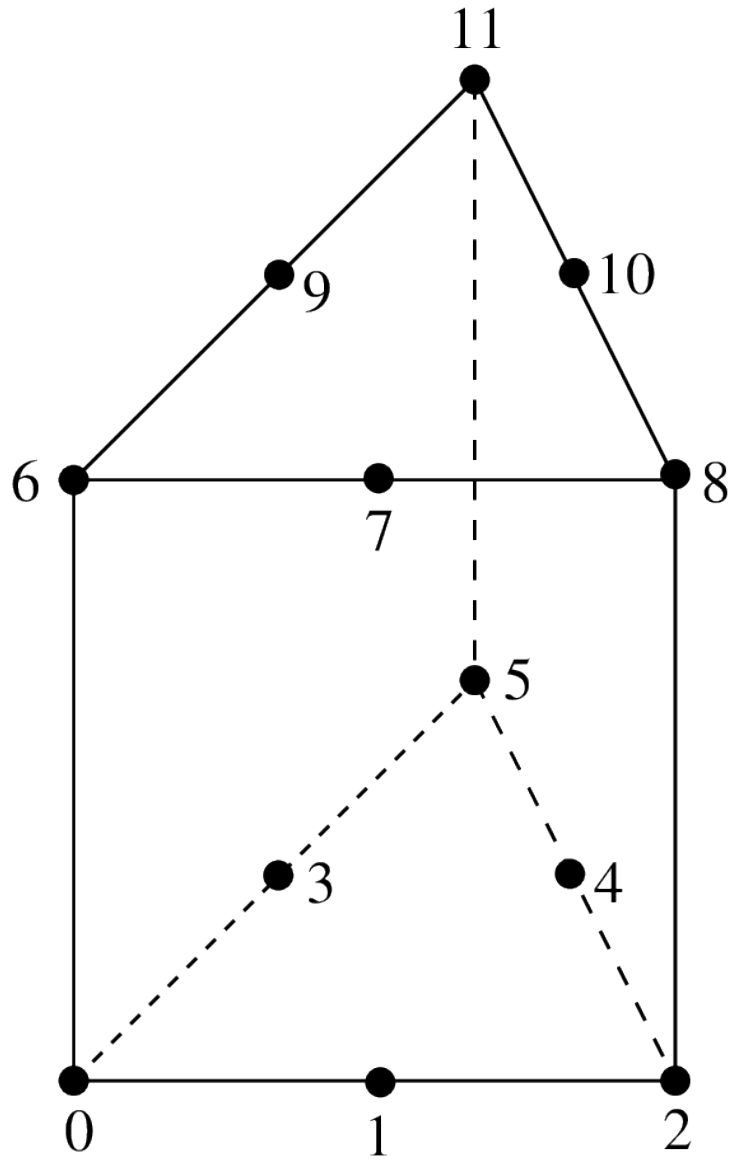


図 27.2 $P_2 \otimes P_1$ プリズム上の Lagrange 要素, 12 自由度, C^0

表 27.10 . プリズム上の Lagrange 要素 “FEM_PRODUCT(FEM_PK(P-1, K1), FEM_PK(1, K2))”

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
$K_1 + K_2,$ $0 \leq K_1, K_2 \leq 255$	$P, 2 \leq P \leq 255$	$(K_2 + 1) \times \frac{(K_1 + P - 1)!}{K_1!(P - 1)!}$	C^0	いいえ ($Q = 1$)	はい ($M = Id$)	はい

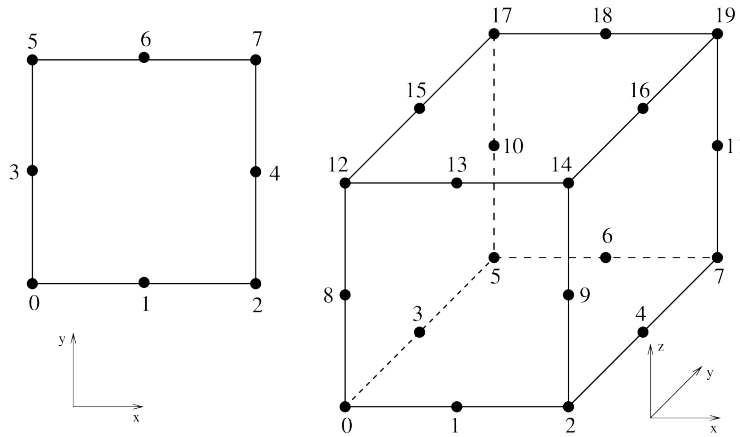


図 27.3 2次元および3次元の不完全 Q_2 要素、8または20自由度、 C^0

表 27.11 不完全 平行四辺形の Lagrange 要素 (Quad 8 と Hexa 20 の Serendipity 要素) "FEM_Q2_INCOMPLETE(P)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	$P, 2 \leq P \leq 3$	8 for $P = 2$ 20 for $P = 3$	C^0	いいえ ($Q = 1$)	はい ($M = Id$)	はい

27.3 ハイアラーキ基底の要素

ハイアラーキ基底の背後にあるアイデアは、異なるレベルの解の記述です：粗いレベル、より洗練されたレベル... 既往のな離散化では、いくつかの自由度は大まかな記述を表し、他のいくつかはより細かな定義などを表します。これは、離散化の余分な空間に対応します。ハイアラーキ基底には、これらの空間のそれぞれの基底が含まれています（これは、メッシュが洗練されている場合、古典的な Lagrange 要素では成り立ちません）。

この手法の利点は、剛性マトリックスの条件数を大幅に向上させることができ、局所的な結果の取得とマルチグ

リッドアプローチによる分解が可能であることです。

27.3.1 次数に関するハイアラーキ的要素

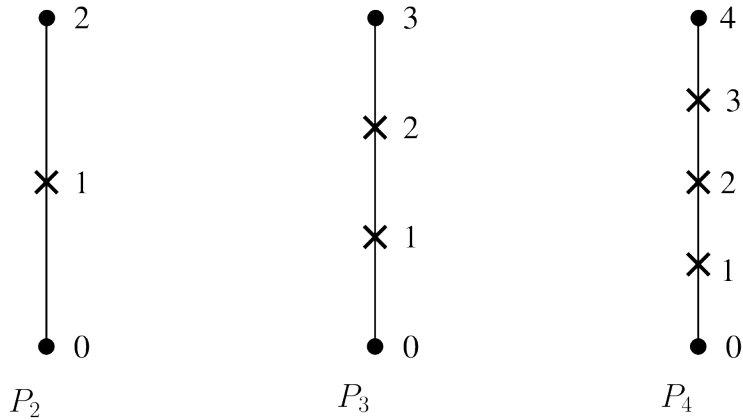


図 27.4 P_K セグメント上のハイアラーキ的要素、 C^0

表 27.12 . シンプレックス上の古典的な Lagrange 要素、"FEM_PK_HIERARCHICAL(P,K)" の次数に関するハイアラーキ基底

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
$K, 0 \leq K \leq 255$	$P, 1 \leq P \leq 255$	$\frac{(K+P)!}{K!P!}$	C^0	いいえ ($Q = 1$)	はい ($M = Id$)	はい

表 27.13 . パラレルパイプ上の古典的な Lagrange 要素、"FEM_QK_HIERARCHICAL(P,K)" の次数に関するハイアラーキ基底

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
$K, 0 \leq K \leq 255$	$P, 1 \leq P \leq 255$	$(K+1)^P$	C^0	いいえ ($Q = 1$)	はい ($M = Id$)	はい

表 27.14 . プリズム上の古典的な Lagrange 要素、"FEM_PK_PRISM_HIERARCHICAL(P,K)" の次数に関するハイアラーキ基底

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
$K, 0 \leq K \leq 255$	$P, 2 \leq P \leq 255$	$(K + 1) \times \frac{(K + P - 1)!}{K!(P - 1)!}$	C^0	いいえ ($Q = 1$)	はい ($M = Id$)	はい

いくつかの特別な選択: P_4 は P_1 の基底をもとに構築されます、追加として P_2 と P_4 の基底も使用されます。

P_6 は P_1 の基底をもとにして構築されます、追加の基底 P_2 と P_6 の基底も使用されます。(P_1 は追加の基底ではありません、これは P_3 の追加基底で P_6 の追加基底です。次の文字で構築することができます : "FEM_GEN_HIERARCHICAL(a, b) ")

27.3.2 複合要素

複合要素の主な関心事は、ハイアラーキ要素を構築することです。しかし、このツールは、区分多項式要素を構築するためにも使用できます。

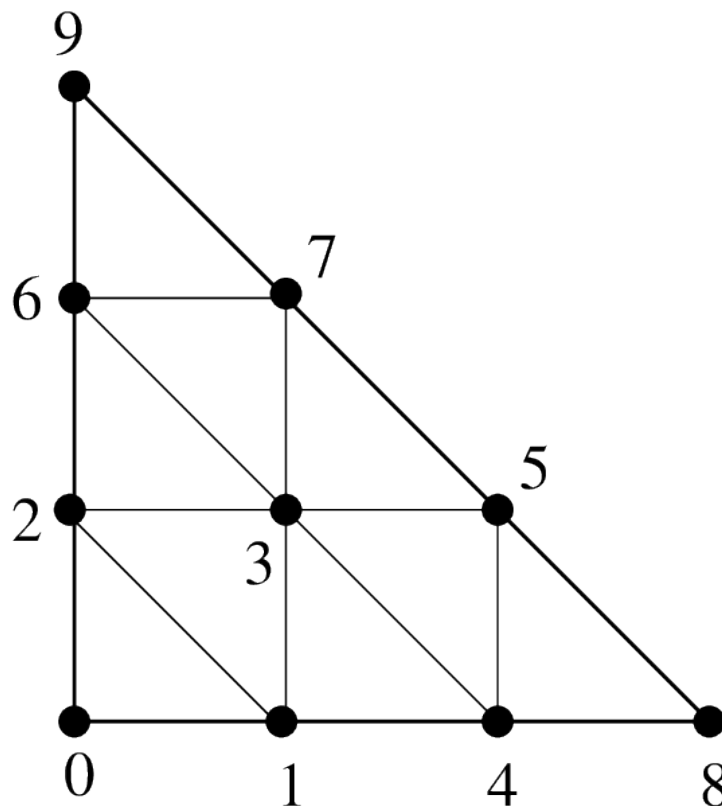


図 27.5 複合要素 "FEM_STRUCTURED_COMPOSITE (FEM_PK (2, 1), 3) "

表 27.15 S のサブ区分を持つ要素に対する有限要素法の構成
"FEM_STRUCTURED_COMPOSITE(FEM1, S)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
FEM1 の次数	FEM1 の次元	変数	変数	いいえ ($Q = 1$)	FEM1 が ^a	区分

対応する複合積分法を使用することが重要です。

27.3.3 ハイアラーキ的な複合要素

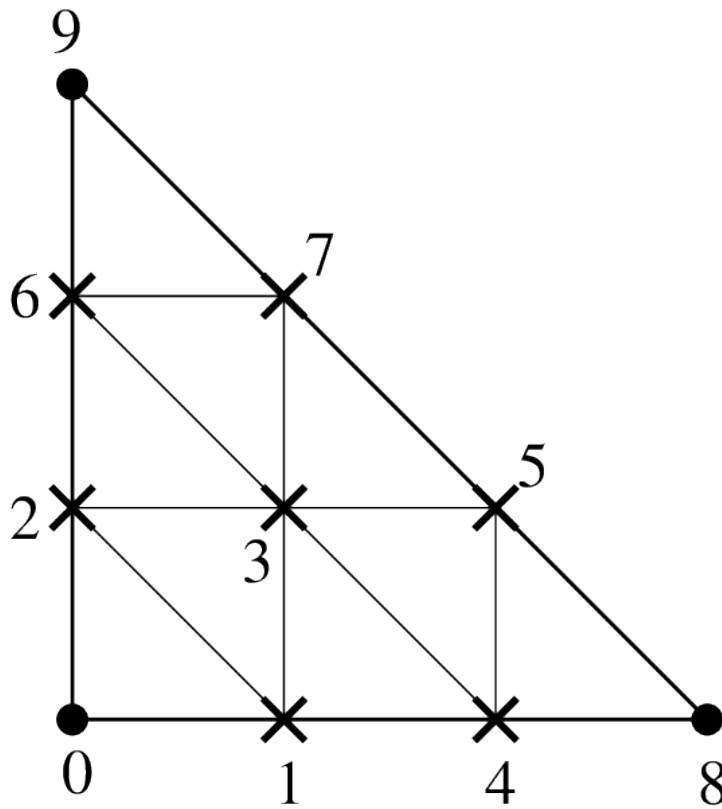


図 27.6 ハイアラーキ的複合要素 "FEM_PK_HIERARCHICAL_COMPOSITE(2, 1, 3)"

表 27.16 S の細分を持つシンプレックス上の有限要素法のハイアラーキ的構成
"FEM_PK_HIERARCHICAL_COMPOSITE(P,K,S)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
K	P	$\frac{(SK+P)!}{(SK)!P!}$	変数	いいえ ($Q = 1$)	はい ($M = Id$)	区分

表 27.17 細分 S のシンプレックス上の有限要素法のハイアラーキ的構成
"FEM_PK_FULL_HIERARCHICAL_COMPOSITE(P,K,S)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
K	P	$\frac{(SK+P)!}{(SK)!P!}$	変数	いいえ ($Q = 1$)	はい ($M = Id$)	区分

"FEM_GEN_HIERARCHICAL (FEM1, FEM2)" と "FEM_STRUCTURED_COMPOSITE (FEM1, S)" により他の構造も可能です。

対応する複合積分法を使用することが重要です。

27.4 古典的なベクトル要素

27.4.1 最小次数要素の Raviart-Thomas

表 27.18 シンプレックスの最下次数要素の Raviart-Thomas "FEM_RT0(P)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
1	P	$P + 1$	H(div)	はい ($Q = P$)	いいえ	はい

表 27.19 平行四辺形 (四角形、6 面体) 上の最小次数要素の Raviart-Thomas "FEM_RT0Q(P)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
1	P	$2P$	H(div)	はい ($Q = P$)	いいえ	はい

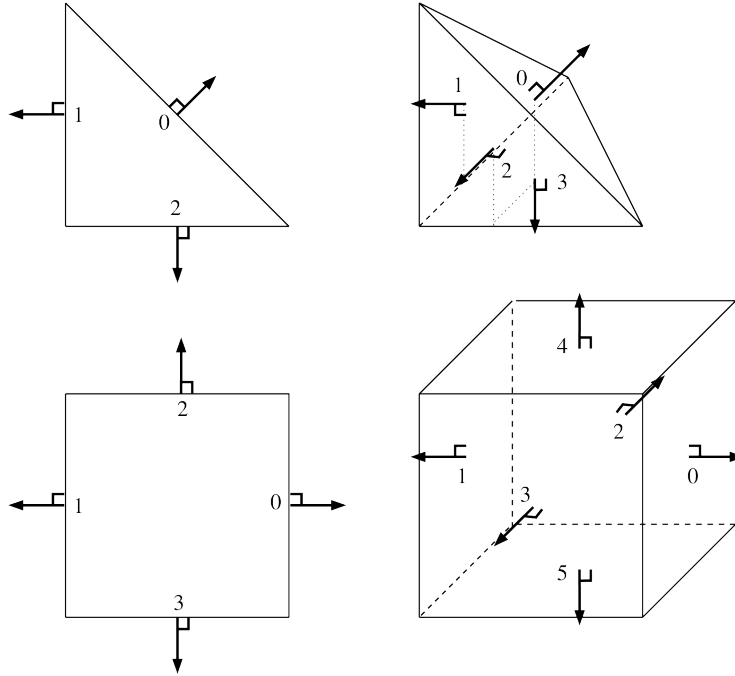


図 27.7 次元 2 と 3 の RT0 要素。(P+1 自由度、H(div))

27.4.2 Nedelec (または Whitney) エッジ要素

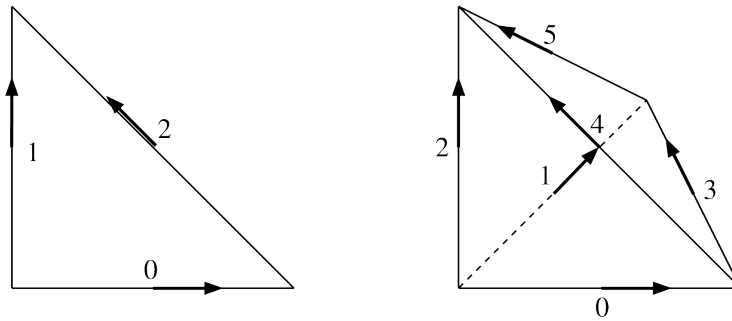


図 27.8 次元 2 と 3 の Nedelec エッジ要素。(P(P+1)/2 dof, H(rot))

表 27.20 Nedelec (または Whitney) のエッジ要素 "FEM_NEDELEC(P)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
1	P	$P(P+1)/2$	H(rot)	はい ($Q = P$)	いいえ	はい

27.5 次元 1 の特定の要素

27.5.1 GaussLobatto 要素

1D GaussLobatto P_K 要素は、セグメント上の古典的な有限要素法 P_K に似ていますが、節点は Gauss-Lobatto-Legendre 直交次数 $2K - 1$ の規則で与えられます。この FEM は、より良好な調整された線形システムにつながる事が知られており、対応する直角位相を使用して（セグメントまたは平行 6 面体に）質量分布を形成することができます。

多項式の係数は Maple であらかじめ計算されています（これらの係数は不調和システムのインバージョンを必要とします）。したがって、以下の値のみに利用できます $K: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 24, 32$ 。 $K = 1$ と $K = 2$ に対しては、古典的な有限要素法 $P1$ と $P2$ であることに注意してください。

表 27.21 セグメント上の GaussLobatto 要素
"FEM_PK_GAUSSLOBATTO1D(K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
K	1	$K + 1$	C^0	いいえ ($Q = 1$)	はい	はい

27.5.2 Hermite 要素

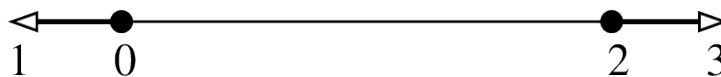


図 27.9 セグメント上の Hermite 要素 P_3 、4 自由度、 C^1

参照要素の基底関数

$$\begin{aligned} \hat{\varphi}_0 &= (2x + 1)(x - 1)^2, & \hat{\varphi}_1 &= x(x - 1)^2, \\ \hat{\varphi}_2 &= x^2(3 - 2x), & \hat{\varphi}_3 &= x^2(x - 1). \end{aligned}$$

この要素は、 τ 等価に近いですが、そうではありません。実際の要素では、頂点の勾配の値に幾何学的変換の勾配が乗算されます。行列 M は単位行列ではありませんが、依然として対角です。

表 27.22 セグメント "FEM_HERMITE(1)" 上の Hermite 要素

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	1	4	C^1	いいえ ($Q = 1$)	いいえ	はい

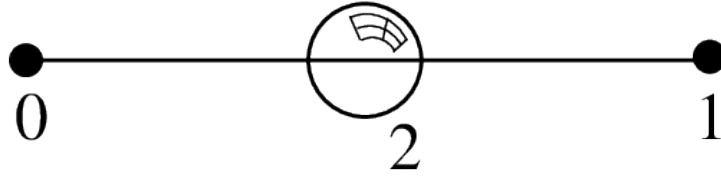


図 27.10 内部気泡関数が追加されたセグメントの P_1 Lagrange 要素、3 自由度、 C^0

27.5.3 追加の気泡関数を備えた Lagrange 要素

表 27.23 内部の気泡関数が追加された Lagrange 要素
"FEM_PK_WITH_CUBIC_BUBBLE(1, 1)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
2	1	3	C^0	いいえ ($Q = 1$)	はい	はい

27.6 次元 2 の特定の要素

27.6.1 追加の気泡関数を持つ要素

表 27.24 追加の内部気泡関数を持つ 3 角形上の Lagrange 要素

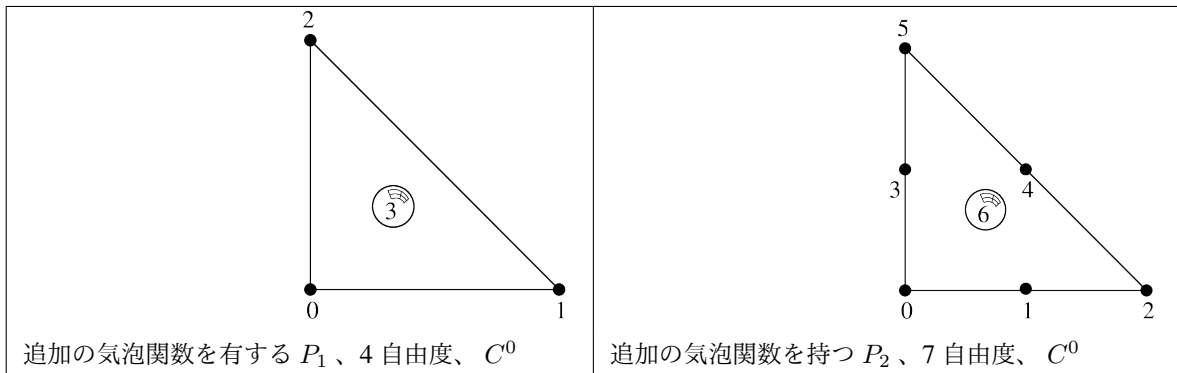


表 27.25 追加の内部気泡関数を持つ Lagrange 要素
"FEM_PK_WITH_CUBIC_BUBBLE(2, K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	2	4 または 7	C^0	いいえ ($Q = 1$)	はい	はい

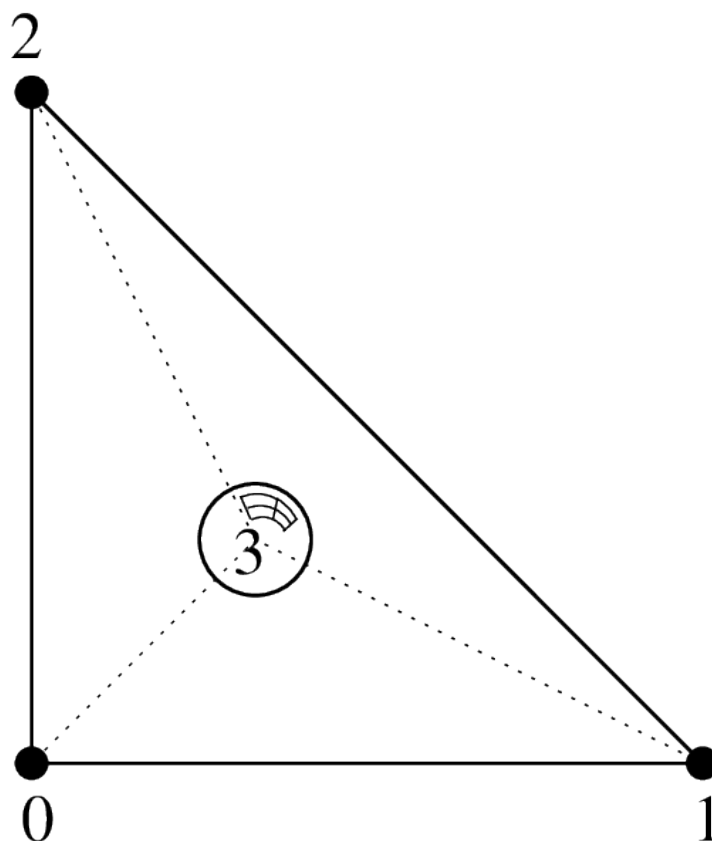


図 27.11 Lagrange 要素 P_1 は、内部区分線形気泡関数を追加した 3 角形上にあります

表 27.26 Lagrange と内部の区分線形気泡関数の追加
"FEM_P1_PIECEWISE_LINEAR_BUBBLE"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
1	2	4 または 7	C^0	いいえ ($Q = 1$)	はい	区分

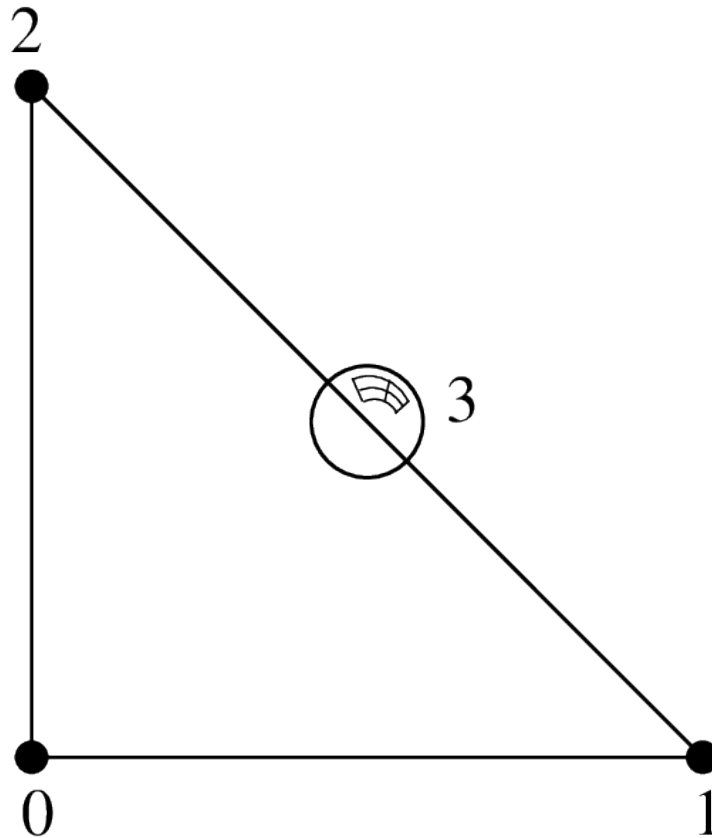


図 27.12 Lagrange 要素 P_1 は、面上に追加の気泡関数を持つ 3 角形上にあります、自由度 4、 C^0

表 27.27 面 0 に気泡関数を追加した Lagrange 要素
"FEM_P1_BUBBLE_FACE(2)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
2	2	4	C^0	いいえ ($Q = 1$)	はい	はい

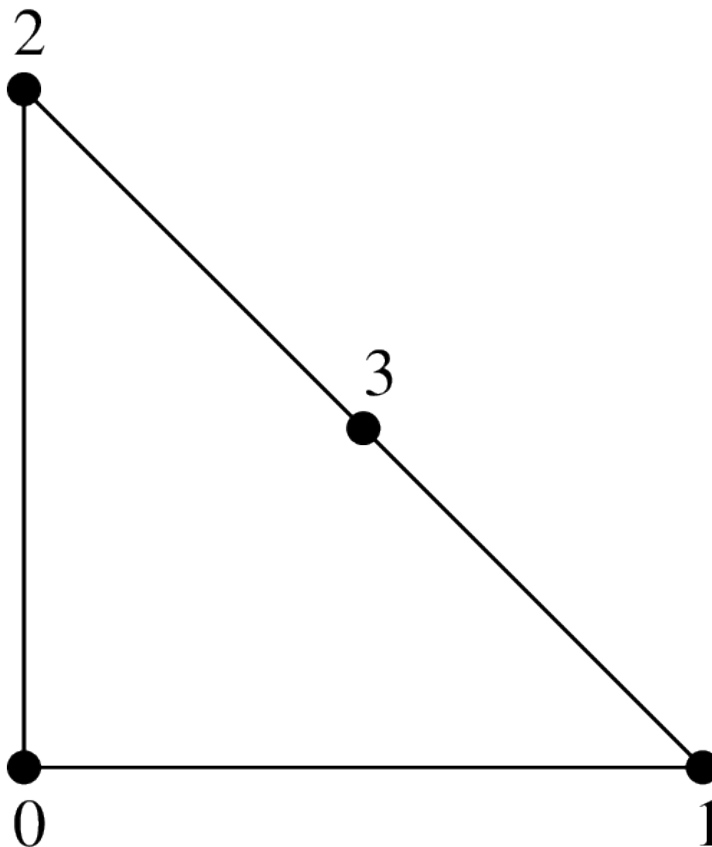


図 27.13 Lagrange 要素 P_1 は、面 0, 4 自由度, C^0 に自由度を追加した 3 角形上にあります。

表 27.28 . 面 0 上の自由度が追加された 3 角形の Lagrange 要素 "FEM_P1_BUBBLE_FACE_LAG"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
2	2	4	C^0	いいえ ($Q = 1$)	はい	はい

27.6.2 不適合 P_1 要素

表 27.29 . 3 角形の非適合要素 "FEM_P1_NONCONFORMING"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
1	2	3	<i>discontinuous</i>	いいえ ($Q = 1$)	はい	はい

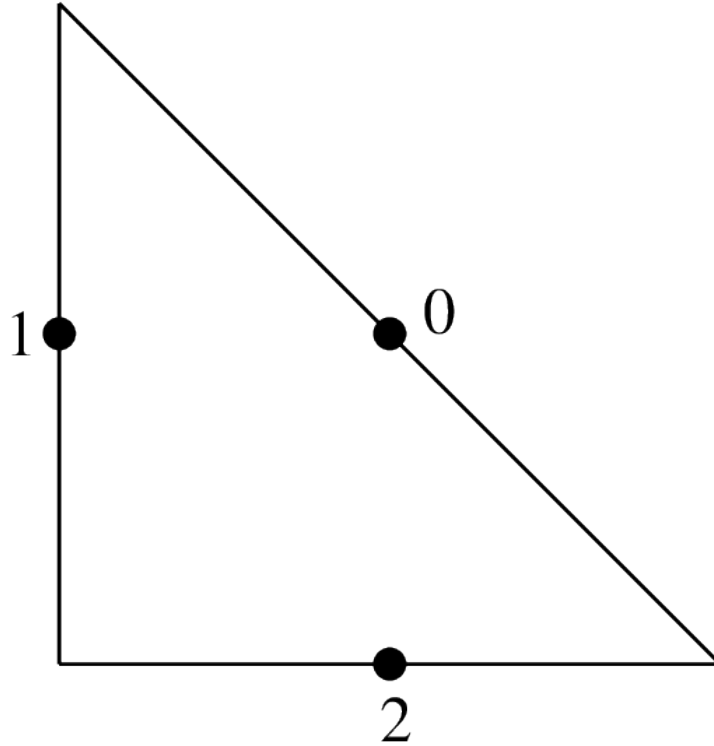


図 27.14 3 角形上の非適合要素 P_1 、3 自由度、不連続

27.6.3 Hermite 要素

参照要素の基底関数：

$$\begin{aligned}
 \hat{\varphi}_0 &= (1-x-y)(1+x+y-2x^2-2y^2-11xy), & (\hat{\varphi}_0(0,0) &= 1), \\
 \hat{\varphi}_1 &= x(1-x-y)(1-x-2y), & (\partial_x \hat{\varphi}_1(0,0) &= 1), \\
 \hat{\varphi}_2 &= y(1-x-y)(1-2x-y), & (\partial_y \hat{\varphi}_2(0,0) &= 1), \\
 \hat{\varphi}_3 &= -2x^3+7x^2y+7xy^2+3x^2-7xy, & (\hat{\varphi}_3(1,0) &= 1), \\
 \hat{\varphi}_4 &= x^3-2x^2y-2xy^2-x^2+2xy, & (\partial_x \hat{\varphi}_4(1,0) &= 1), \\
 \hat{\varphi}_5 &= xy(y+2x-1), & (\partial_y \hat{\varphi}_5(1,0) &= 1), \\
 \hat{\varphi}_6 &= 7x^2y+7xy^2-2y^3+3y^2-7xy, & (\hat{\varphi}_6(0,1) &= 1), \\
 \hat{\varphi}_7 &= xy(x+2y-1), & (\partial_x \hat{\varphi}_7(0,1) &= 1), \\
 \hat{\varphi}_8 &= y^3-2x^2y-2xy^2-y^2+2xy, & (\partial_y \hat{\varphi}_8(0,1) &= 1), \\
 \hat{\varphi}_9 &= 27xy(1-x-y), & (\hat{\varphi}_9(1/3,1/3) &= 1),
 \end{aligned}$$

この要素は τ 等価ではありません (行列 M は単位行列ではありません)。実際の要素では $\hat{\varphi}_4$ と $\hat{\varphi}_7$ を使って対応する頂点の勾配を一致させます。 ($\hat{\varphi}_5, \hat{\varphi}_8$) と ($\hat{\varphi}_6, \hat{\varphi}_9$) を 2 つの他の頂点に対して使用します。

表 27.30 3 角形の Hermite 要素 "FEM_HERMITE(2)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	2	10	C^0	いいえ ($Q = 1$)	いいえ	はい

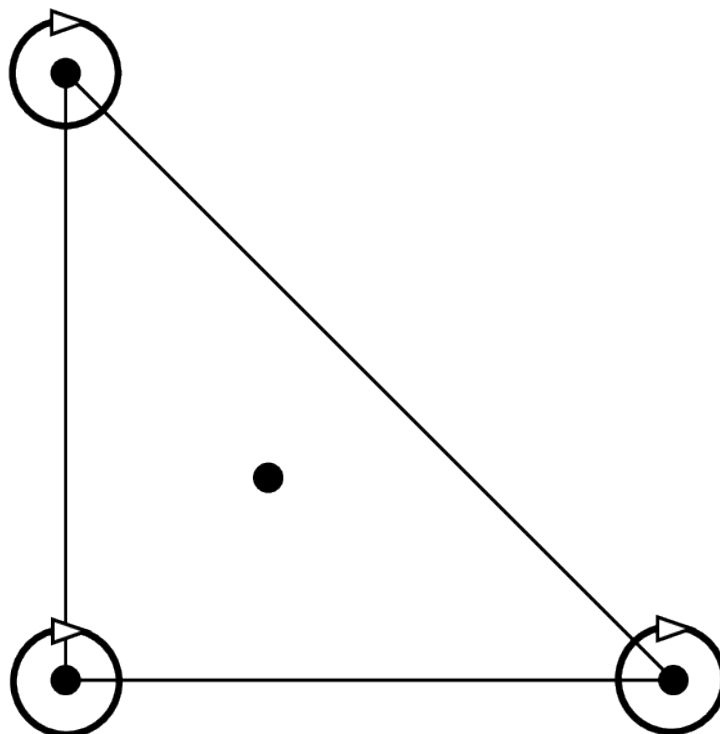


図 27.15 3 角形の Hermite 要素, P_3 , 10 自由度, C^0

27.6.4 Morley 要素

この要素は τ 等価 (行列 M は単位行列ではありません) ではありません。特に、4 次問題の非適合離散化には、 C^1 ではないにもかかわらず使用されます。

表 27.31 3 角形上の Morley 要素 "FEM_MORLEY"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
2	2	6	不連続な	いいえ ($Q = 1$)	いいえ	はい

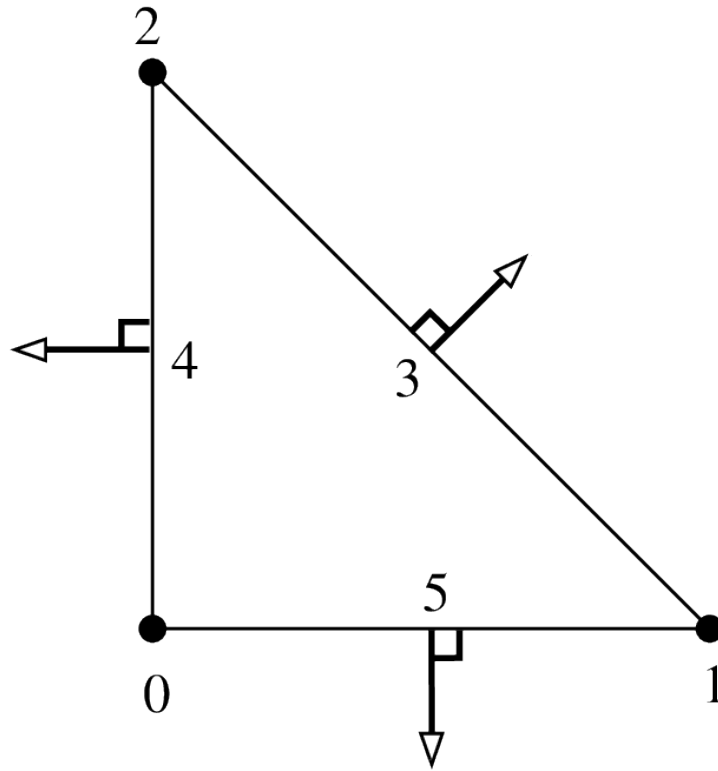


図 27.16 3 角形 Morley 要素、 P_2 、6 自由度、 C^0

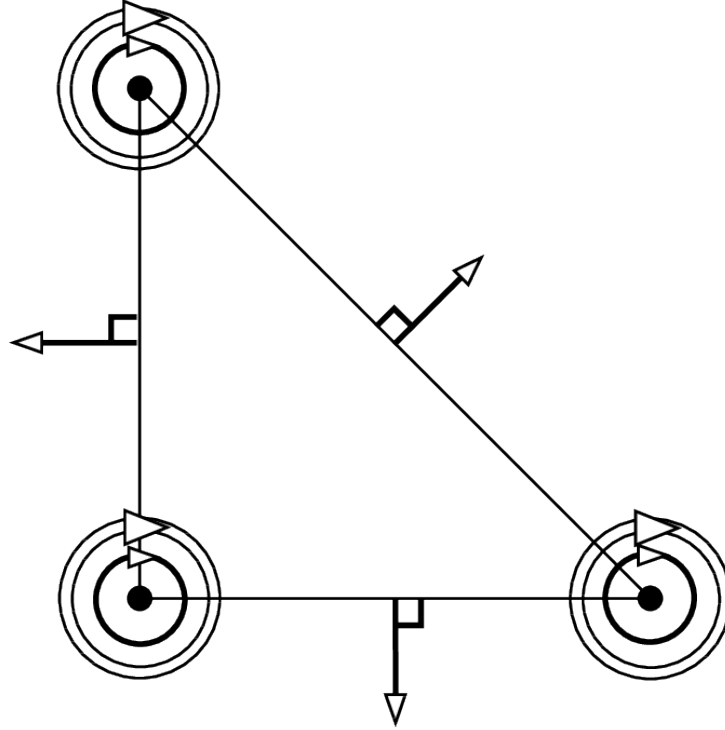


図 27.17 Argyris 要素、 P_5 , 21 自由度, C^1

27.6.5 Argyris 要素

参照要素の基底関数は次のとおりです:

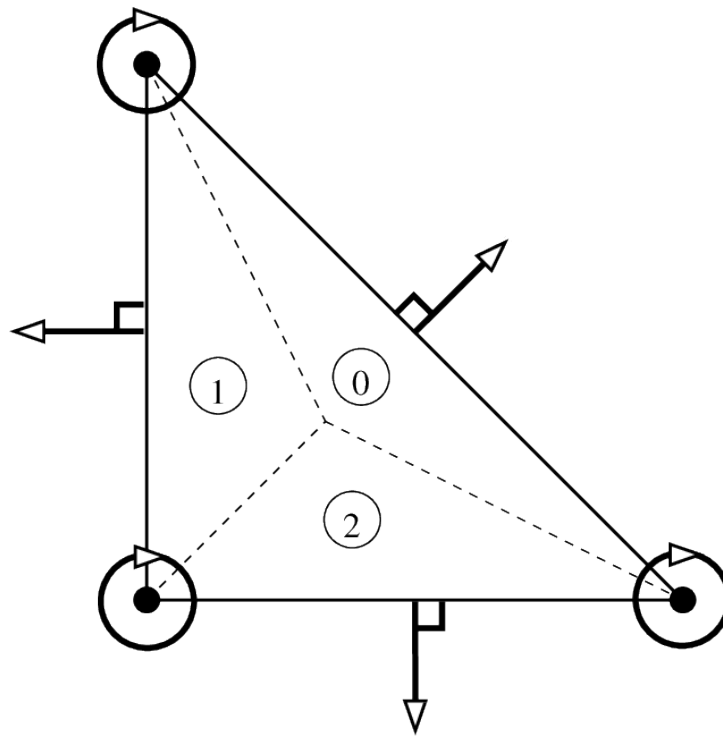
$$\begin{aligned}
 \hat{\varphi}_0(x, y) &= 1 - 10x^3 - 10y^3 + 15x^4 - 30x^2y^2 + 15y^4 - 6x^5 + 30x^3y^2 + 30x^2y^3 - 6y^5, & (\hat{\varphi}_0(0, 0) = 1), \\
 \hat{\varphi}_1(x, y) &= x - 6x^3 - 11xy^2 + 8x^4 + 10x^2y^2 + 18xy^3 - 3x^5 + x^3y^2 - 10x^2y^3 - 8xy^4, & (\partial_x \hat{\varphi}_1(0, 0) = 1), \\
 \hat{\varphi}_2(x, y) &= y - 11x^2y - 6y^3 + 18x^3y + 10x^2y^2 + 8y^4 - 8x^4y - 10x^3y^2 + x^2y^3 - 3y^5, & (\partial_y \hat{\varphi}_2(0, 0) = 1), \\
 \hat{\varphi}_3(x, y) &= 0.5x^2 - 1.5x^3 + 1.5x^4 - 1.5x^2y^2 - 0.5x^5 + 1.5x^3y^2 + x^2y^3, & (\partial_{xx}^2 \hat{\varphi}_3(0, 0) = 1), \\
 \hat{\varphi}_4(x, y) &= xy - 4x^2y - 4xy^2 + 5x^3y + 10x^2y^2 + 5xy^3 - 2x^4y - 6x^3y^2 - 6x^2y^3 - 2xy^4, & (\partial_{xy}^2 \hat{\varphi}_4(0, 0) = 1), \\
 \hat{\varphi}_5(x, y) &= 0.5y^2 - 1.5y^3 - 1.5x^2y^2 + 1.5y^4 + x^3y^2 + 1.5x^2y^3 - 0.5y^5, & (\partial_{yy}^2 \hat{\varphi}_5(0, 0) = 1), \\
 \hat{\varphi}_6(x, y) &= 10x^3 - 15x^4 + 15x^2y^2 + 6x^5 - 15x^3y^2 - 15x^2y^3, & (\hat{\varphi}_6(1, 0) = 1), \\
 \hat{\varphi}_7(x, y) &= -4x^3 + 7x^4 - 3.5x^2y^2 - 3x^5 + 3.5x^3y^2 + 3.5x^2y^3, & (\partial_x \hat{\varphi}_7(1, 0) = 1), \\
 \hat{\varphi}_8(x, y) &= -5x^2y + 14x^3y + 18.5x^2y^2 - 8x^4y - 18.5x^3y^2 - 13.5x^2y^3, & (\partial_y \hat{\varphi}_8(1, 0) = 1), \\
 \hat{\varphi}_9(x, y) &= 0.5x^3 - x^4 + 0.25x^2y^2 + 0.5x^5 - 0.25x^3y^2 - 0.25x^2y^3, & (\partial_{xx}^2 \hat{\varphi}_9(1, 0) = 1), \\
 \hat{\varphi}_{10}(x, y) &= x^2y - 3x^3y - 3.5x^2y^2 + 2x^4y + 3.5x^3y^2 + 2.5x^2y^3, & (\partial_{xy}^2 \hat{\varphi}_{10}(1, 0) = 1), \\
 \hat{\varphi}_{11}(x, y) &= 1.25x^2y^2 - 0.75x^3y^2 - 1.25x^2y^3, & (\partial_{yy}^2 \hat{\varphi}_{11}(1, 0) = 1), \\
 \hat{\varphi}_{12}(x, y) &= 10y^3 + 15x^2y^2 - 15y^4 - 15x^3y^2 - 15x^2y^3 + 6y^5, & (\hat{\varphi}_{12}(0, 1) = 1), \\
 \hat{\varphi}_{13}(x, y) &= -5xy^2 + 18.5x^2y^2 + 14xy^3 - 13.5x^3y^2 - 18.5x^2y^3 - 8xy^4, & (\partial_x \hat{\varphi}_{13}(0, 1) = 1), \\
 \hat{\varphi}_{14}(x, y) &= -4y^3 - 3.5x^2y^2 + 7y^4 + 3.5x^3y^2 + 3.5x^2y^3 - 3y^5, & (\partial_y \hat{\varphi}_{14}(0, 0) = 1), \\
 \hat{\varphi}_{15}(x, y) &= 1.25x^2y^2 - 1.25x^3y^2 - 0.75x^2y^3, & (\partial_{xx}^2 \hat{\varphi}_{15}(0, 1) = 1), \\
 \hat{\varphi}_{16}(x, y) &= xy^2 - 3.5x^2y^2 - 3xy^3 + 2.5x^3y^2 + 3.5x^2y^3 + 2xy^4, & (\partial_{xy}^2 \hat{\varphi}_{16}(0, 1) = 1), \\
 \hat{\varphi}_{17}(x, y) &= 0.5y^3 + 0.25x^2y^2 - y^4 - 0.25x^3y^2 - 0.25x^2y^3 + 0.5y^5, & (\partial_{yy}^2 \hat{\varphi}_{17}(0, 1) = 1), \\
 \hat{\varphi}_{18}(x, y) &= \sqrt{2}(-8x^2y^2 + 8x^3y^2 + 8x^2y^3), & (\sqrt{0.5}(\partial_x \hat{\varphi}_{18}(0.5, 0.5) + \partial_y \hat{\varphi}_{18}(0.5, 0.5)) = 1), \\
 \hat{\varphi}_{19}(x, y) &= -16xy^2 + 32x^2y^2 + 32xy^3 - 16x^3y^2 - 32x^2y^3 - 16xy^4, & (-\partial_x \hat{\varphi}_{19}(0, 0.5) = 1), \\
 \hat{\varphi}_{20}(x, y) &= -16x^2y + 32x^3y + 32x^2y^2 - 16x^4y - 32x^3y^2 - 16x^2y^3, & (-\partial_y \hat{\varphi}_{20}(0.5, 0) = 1),
 \end{aligned}$$

この要素は τ 等価ではありません (行列 M は単位行列ではありません)。実際の要素では、変換された基底関数の線形結合 $\hat{\varphi}_i$ を使用して、勾配、2次微分、および正規微分を面に一致させます。行列 M を使用すると、非線形幾何変換 (例えば、曲線境界) であっても、Argyris 要素を定義することができます。

表 27.32 3 角形の Argyris 要素 "FEM_ARGYRIS"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
5	2	21	C^1	いいえ ($Q = 1$)	いいえ	はい

27.6.6 Hsieh-Clough-Tocher 要素

図 27.18 Hsieh-Clough-Tocher (HCT) 要素, P_3 , 12 自由度, C^1

この要素は τ 等価ではありません。これは複合要素です。3つのサブ3角形のそれぞれについて3次の多項式 (図 *Hsieh-Clough-Tocher (HCT) 要素*, 12 自由度, と [ciarlet1978] を参照)。この有限要素には "IM_HCT_COMPOSITE" 積分メソッドを使うことを強く推奨します。自由度の値は次のとおりです。最初の2番目と3番目の頂点のLagrange自由度は0,3,6です。第1の変数に対する導関数の第1、第4、第7、第2の変数に対する導関数は2,5,8であり、面0,1,2のそれぞれの正規導関数については9,10,11です。

表 27.33 三角 "FEM_HCT_TRIANGLE" 上の HCT 要素

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	2	12	C^1	いいえ ($Q = 1$)	いいえ	区分

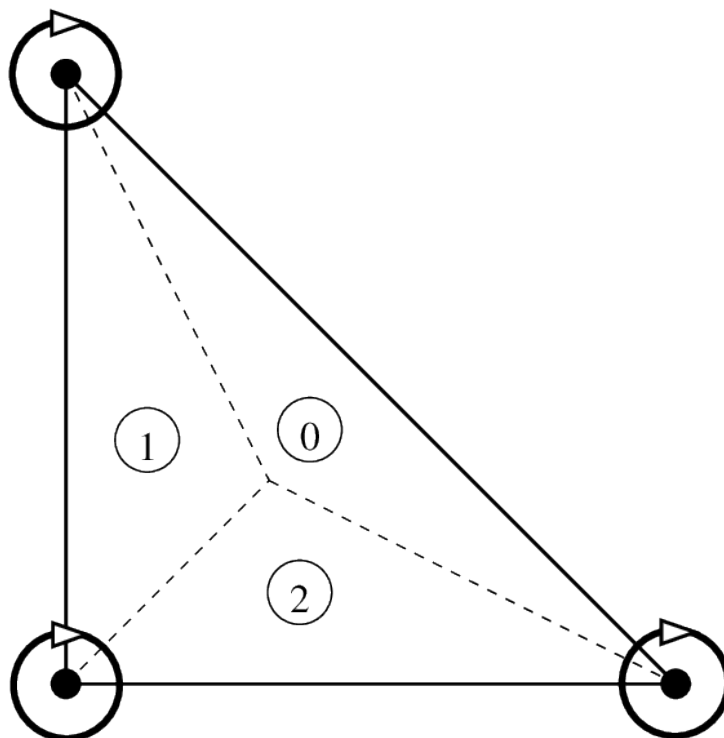


図 27.19 縮小 Hsieh-Clough-Tocher (縮小 HCT) 要素、 P_3 , 9 自由度, C^1

この要素は縮小形でも存在し、正規分布は各辺で次数 1 の多項式と仮定されます (図 縮小 Hsieh-Clough-Tocher (縮小 HCT) 要素、, 9 自由度, を参照)。

表 27.34 三角 "FEM_REDUCED_HCT_TRIANGLE" 上の縮小 HCT 要素

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	2	9	C^1	いいえ ($Q = 1$)	いいえ	区分

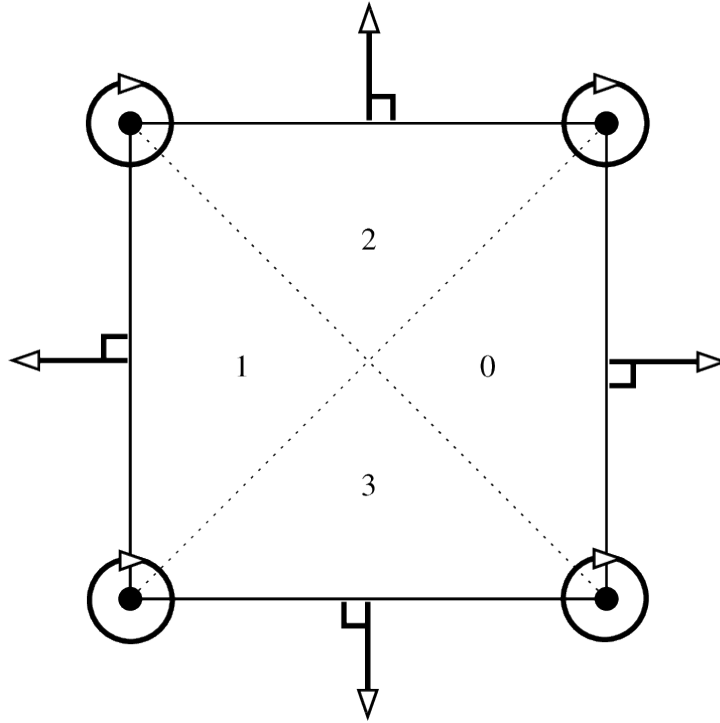


図 27.20 分割された四辺形の複合要素 P_3 , 16 自由度, C^1

27.6.7 四角形上の複合 C^1 要素

この要素は τ 等価ではありません。これは複合要素です。4 つのサブ 3 角形のそれぞれの次数 3 の多項式です (分割された四辺形の複合要素, 16 自由度, を参照)。Fraeijs de Veubeke-Sander 要素に対応します ([ciarlet1978] を参照)。この有限要素では "IM_QUADC1_COMPOSITE" 積分法を使うことを強く推奨します。

表 27.35 . 四角形 (FVS) 上にある 複合要素 複合要素
"FEM_QUADC1_COMPOSITE"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	2	16	C^1	いいえ ($Q = 1$)	いいえ	区分

この要素は、その縮小形でも存在し、通常の導関数は各辺で次数 1 の多項式と仮定されます (図 四辺形の縮小複合要素、区分的、12 自由度、 を参照)

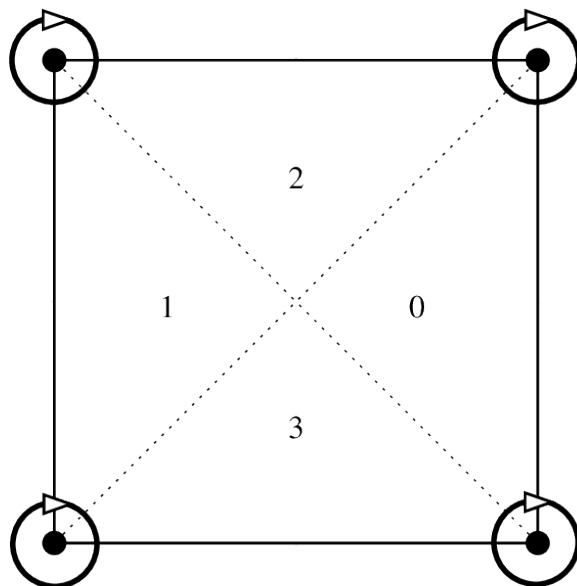


図 27.21 四辺形の縮小複合要素、区分的 P_3 、12 自由度、 C^1

表 27.36 四角形 (縮小 FVS) の縮小複合要素
"FEM_REDUCED_QUADC1_COMPOSITE"

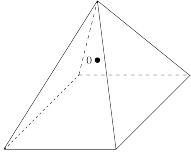
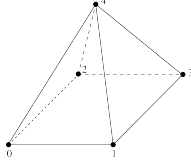
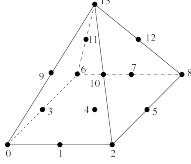
度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	2	12	C^1	いいえ ($Q = 1$)	いいえ	区分

27.7 次元 3 の特定の要素

27.7.1 3D ピラミッド上の Lagrange 要素

[GR-GH1999] および [BE-CO-DU2010] に基づいて、*GetFEM++* は 0,1 次および 2 次のいくつかの Lagrange ピラミッド要素を提案します。詳細については、これらの参考文献を参照してください。提案された要素は、3 角形面の標準 P_1 または P_2 Lagrange 有限要素法と四角形面上の標準 Lagrange 有限要素法 Q_1 または Q_2 です。

表 27.37 0次、1次、2次のピラミッド型要素上の Lagrange 要素

 <p>自由度 1 の 0 次ピラミッド型要素</p>	 <p>自由度 5 の 1 次ピラミッド型要素</p>	 <p>自由度 14 の 2 次ピラミッド型要素</p>
--	--	---

関連する幾何変換は、 $K = 1$ または 2 の場合の "GT_PYRAMID(K)" です。関連する積分法は "IM_PYRAMID(im)" です。ここで im は 6 面体の積分方法です。(あるいは積分法 "IM_PYRAMID_COMPOSITE(im)" です。ここで im は 4 面体の積分方法ですが、理論的にはそれほど正確ではありません。) 形状関数は多項式ではなく有理数です。最初の次数については、形状関数が読み取られます。

$$\begin{aligned} \hat{\varphi}_0(x, y, z) &= \frac{1}{4} \left(1 - x - y - z + \frac{xy}{1-z} \right), \\ \hat{\varphi}_1(x, y, z) &= \frac{1}{4} \left(1 + x - y - z - \frac{xy}{1-z} \right), \\ \hat{\varphi}_2(x, y, z) &= \frac{1}{4} \left(1 - x + y - z - \frac{xy}{1-z} \right), \\ \hat{\varphi}_3(x, y, z) &= \frac{1}{4} \left(1 + x + y - z + \frac{xy}{1-z} \right), \\ \hat{\varphi}_4(x, y, z) &= z. \end{aligned}$$

2 次のために、次式を設定します

$$\xi_0 = \frac{1-z-x}{2}, \quad \xi_1 = \frac{1-z-y}{2}, \quad \xi_2 = \frac{1-z+x}{2}, \quad \xi_3 = \frac{1-z+y}{2}, \quad \xi_4 = z,$$

形状関数の読み込み :

$$\begin{aligned} \hat{\varphi}_0(x, y, z) &= \frac{\xi_0 \xi_1}{(1 - \xi_4)^2} ((1 - \xi_4 - 2\xi_0)(1 - \xi_4 - 2\xi_1) - \xi_4(1 - \xi_4)), \\ \hat{\varphi}_1(x, y, z) &= 4 \frac{\xi_0 \xi_1 \xi_2}{(1 - \xi_4)^2} (2\xi_1 - (1 - \xi_4)), \\ \hat{\varphi}_2(x, y, \xi_4) &= \frac{\xi_1 \xi_2}{(1 - \xi_4)^2} ((1 - \xi_4 - 2\xi_1)(1 - \xi_4 - 2\xi_2) - \xi_4(1 - \xi_4)), \\ \hat{\varphi}_3(x, y, z) &= 4 \frac{\xi_3 \xi_0 \xi_1}{(1 - \xi_4)^2} (2\xi_0 - (1 - \xi_4)), \\ \hat{\varphi}_4(x, y, z) &= 16 \frac{\xi_0 \xi_1 \xi_2 \xi_3}{(1 - \xi_4)^2}, \\ \hat{\varphi}_5(x, y, z) &= 4 \frac{\xi_1 \xi_2 \xi_3}{(1 - \xi_4)^2} (2\xi_2 - (1 - \xi_4)), \\ \hat{\varphi}_6(x, y, z) &= \frac{\xi_3 \xi_0}{(1 - \xi_4)^2} ((1 - \xi_4 - 2\xi_3)(1 - \xi_4 - 2\xi_0) - \xi_4(1 - \xi_4)), \\ \hat{\varphi}_7(x, y, z) &= 4 \frac{\xi_2 \xi_3 \xi_0}{(1 - \xi_4)^2} (2\xi_3 - (1 - \xi_4)), \\ \hat{\varphi}_8(x, y, z) &= \frac{\xi_2 \xi_3}{(1 - \xi_4)^2} ((1 - \xi_4 - 2\xi_2)(1 - \xi_4 - 2\xi_3) - \xi_4(1 - \xi_4)), \\ \hat{\varphi}_9(x, y, z) &= 4 \frac{\xi_4}{1 - \xi_4} \xi_0 \xi_1, \\ \hat{\varphi}_{10}(x, y, z) &= 4 \frac{\xi_4}{1 - \xi_4} \xi_1 \xi_2, \\ \hat{\varphi}_{11}(x, y, z) &= 4 \frac{\xi_4}{1 - \xi_4} \xi_3 \xi_0, \\ \hat{\varphi}_{12}(x, y, z) &= 4 \frac{\xi_4}{1 - \xi_4} \xi_2 \xi_3, \\ \hat{\varphi}_{13}(x, y, z) &= \xi_4(2\xi_4 - 1). \end{aligned}$$

表 27.38 次数 0,1 または 2 の連続 Lagrange 要素
"FEM_PYRAMID_LAGRANGE(K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
0	3	1	不連続な	いいえ ($Q = 1$)	はい	いいえ
1	3	5	C^0	いいえ ($Q = 1$)	はい	いいえ
2	3	14	C^0	いいえ ($Q = 1$)	はい	いいえ

表 27.39 次数 0、1、または 2 の不連続 Lagrange 要素
"FEM_PYRAMID_DISCONTINUOUS_LAGRANGE(K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
0	3	1	不連続な	いいえ ($Q = 1$)	はい	いいえ
1	3	5	不連続な	いいえ ($Q = 1$)	はい	いいえ
2	3	14	不連続な	いいえ ($Q = 1$)	はい	いいえ

27.7.2 追加の気泡関数を持つ要素

表 27.40 追加の内部気泡関数を有する 4 面体上の Lagrange 要素

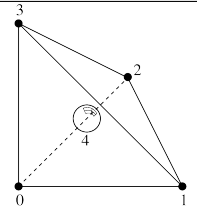
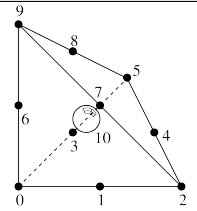
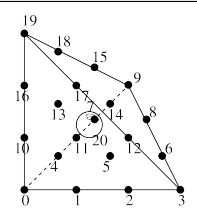
		
追加の気泡関数を持つ P_1 、5 自由度、 C^0	追加の気泡関数を持つ P_2 、11 自由度、 C^0	追加の気泡関数を持つ P_3 、21 自由度、 C^0

表 27.41 追加の内部気泡関数を持つ Lagrange 要素
"FEM_PK_WITH_CUBIC_BUBBLE(3, K)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
4	3	5, 11 または 21	C^0	いいえ ($Q = 1$)	はい	はい

表 27.42 面 0 に気泡関数を追加した Lagrange 要素
"FEM_P1_BUBBLE_FACE(3)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	3	5	C^0	いいえ ($Q = 1$)	はい	はい

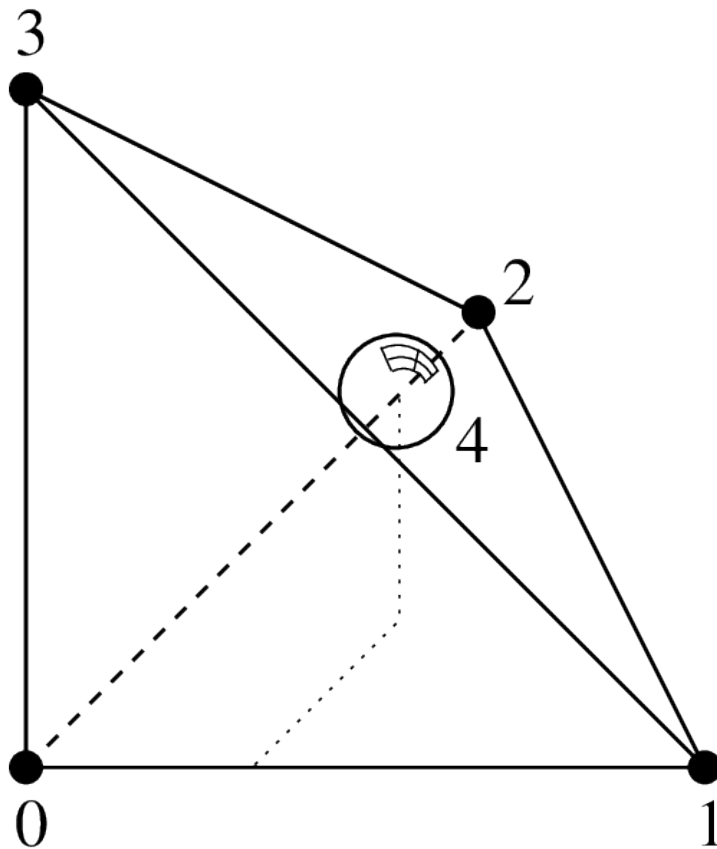


図 27.22 P_1 4 面体上の Lagrange 要素で、面上に追加の気泡関数を持つ、5 自由度、 C^0

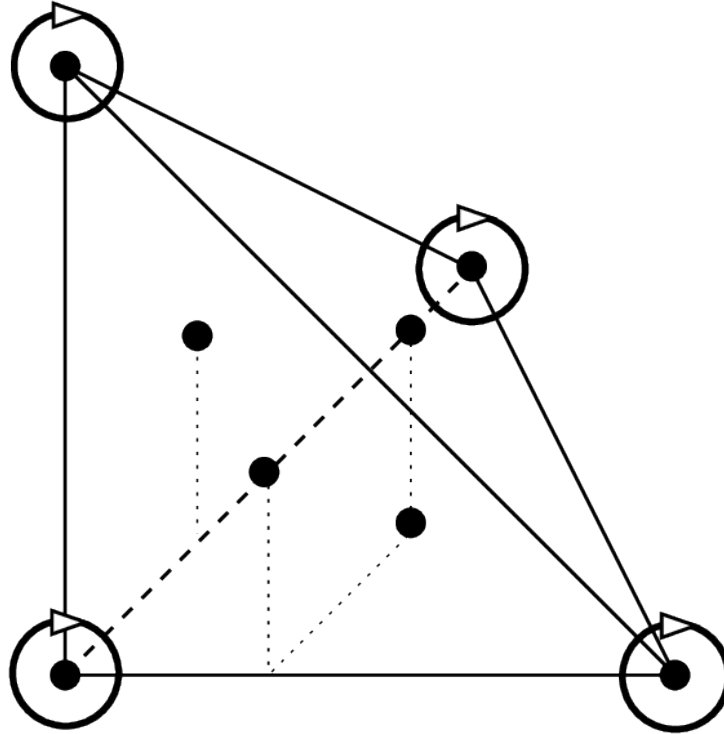


図 27.23 4 面体の Hermite 要素、 P_3 , 20 自由度, C^0

27.7.3 Hermite 要素

参照要素の基底関数：

$$\begin{aligned}
 \hat{\varphi}_0(x, y) &= 1 - 3x^2 - 13xy - 13xz - 3y^2 - 13yz - 3z^2 + 2x^3 + 13x^2y + 13x^2z \\
 &\quad + 13xy^2 + 33xyz + 13xz^2 + 2y^3 + 13y^2z + 13yz^2 + 2z^3, & (\hat{\varphi}_0(0, 0, 0) &= 1), \\
 \hat{\varphi}_1(x, y) &= x - 2x^2 - 3xy - 3xz + x^3 + 3x^2y + 3x^2z + 2xy^2 + 4xyz + 2xz^2, & (\partial_x \hat{\varphi}_1(0, 0, 0) &= 1), \\
 \hat{\varphi}_2(x, y) &= y - 3xy - 2y^2 - 3yz + 2x^2y + 3xy^2 + 4xyz + y^3 + 3y^2z + 2yz^2, & (\partial_y \hat{\varphi}_2(0, 0, 0) &= 1), \\
 \hat{\varphi}_3(x, y) &= z - 3xz - 3yz - 2z^2 + 2x^2z + 4xyz + 3xz^2 + 2y^2z + 3yz^2 + z^3, & (\partial_z \hat{\varphi}_3(0, 0, 0) &= 1), \\
 \hat{\varphi}_4(x, y) &= 3x^2 - 7xy - 7xz - 2x^3 + 7x^2y + 7x^2z + 7xy^2 + 7xyz + 7xz^2, & (\hat{\varphi}_4(1, 0, 0) &= 1), \\
 \hat{\varphi}_5(x, y) &= -x^2 + 2xy + 2xz + x^3 - 2x^2y - 2x^2z - 2xy^2 - 2xyz - 2xz^2, & (\partial_x \hat{\varphi}_5(1, 0, 0) &= 1), \\
 \hat{\varphi}_6(x, y) &= -xy + 2x^2y + xy^2, & (\partial_y \hat{\varphi}_6(1, 0, 0) &= 1), \\
 \hat{\varphi}_7(x, y) &= -xz + 2x^2z + xz^2, & (\partial_z \hat{\varphi}_7(1, 0, 0) &= 1), \\
 \hat{\varphi}_8(x, y) &= -7xy + 3y^2 - 7yz + 7x^2y + 7xy^2 + 7xyz - 2y^3 + 7y^2z + 7yz^2, & (\hat{\varphi}_8(0, 1, 0) &= 1), \\
 \hat{\varphi}_9(x, y) &= -xy + x^2y + 2xy^2, & (\partial_x \hat{\varphi}_9(0, 1, 0) &= 1), \\
 \hat{\varphi}_{10}(x, y) &= 2xy - y^2 + 2yz - 2x^2y - 2xy^2 - 2xyz + y^3 - 2y^2z - 2yz^2, & (\partial_y \hat{\varphi}_{10}(0, 1, 0) &= 1), \\
 \hat{\varphi}_{11}(x, y) &= -yz + 2y^2z + yz^2, & (\partial_z \hat{\varphi}_{11}(0, 1, 0) &= 1), \\
 \hat{\varphi}_{12}(x, y) &= -7xz - 7yz + 3z^2 + 7x^2z + 7xyz + 7xz^2 + 7y^2z + 7yz^2 - 2z^3, & (\hat{\varphi}_{12}(0, 0, 1) &= 1), \\
 \hat{\varphi}_{13}(x, y) &= -xz + x^2z + 2xz^2, & (\partial_x \hat{\varphi}_{13}(0, 0, 1) &= 1), \\
 \hat{\varphi}_{14}(x, y) &= -yz + y^2z + 2yz^2, & (\partial_y \hat{\varphi}_{14}(0, 0, 1) &= 1), \\
 \hat{\varphi}_{15}(x, y) &= 2xz + 2yz - z^2 - 2x^2z - 2xyz - 2xz^2 - 2y^2z - 2yz^2 + z^3, & (\partial_z \hat{\varphi}_{15}(0, 0, 1) &= 1), \\
 \hat{\varphi}_{16}(x, y) &= 27xyz, & (\hat{\varphi}_{16}(1/3, 1/3, 1/3) &= 1), \\
 \hat{\varphi}_{17}(x, y) &= 27yz - 27xyz - 27y^2z - 27yz^2, & (\hat{\varphi}_{17}(0, 1/3, 1/3) &= 1), \\
 \hat{\varphi}_{18}(x, y) &= 27xz - 27x^2z - 27xyz - 27xz^2, & (\hat{\varphi}_{18}(1/3, 0, 1/3) &= 1), \\
 \hat{\varphi}_{19}(x, y) &= 27xy - 27x^2y - 27xy^2 - 27xyz, & (\hat{\varphi}_{19}(1/3, 1/3, 0) &= 1),
 \end{aligned}$$

この要素は τ 等価ではありません (行列 M は単位行列ではありません)。実際の要素では、基底関数の線形結合 $\hat{\varphi}_8$ 、 $\hat{\varphi}_{12}$ と $\hat{\varphi}_{16}$ は、対応する頂点の勾配を一致させるために使用されます。他の頂点のアイテム。

表 27.43 4 面体の Hermite 要素 "FEM_HERMITE(3)"

度	寸法	自由度数	クラス	ベクター	τ 等価	多項式
3	3	20	C^0	いいえ ($Q = 1$)	いいえ	はい

第 28 章

付録 B. 立体求積法のリスト

積分法は 2 種類あります。任意の関数の多項式と近似された積分（立方体式）の正確な積分。正確な積分は、すべての要素が多項式であり、幾何変換が線形である場合にのみ使用できます。

積分法の記述子は、関数:

```
ppi = getfem::int_method_descriptor("name of method");
```

"name of method" は既存のメソッドの中から選択される文字列です。

tests ディレクトリにある integration プログラムは、それぞれの積分法の次数を列挙して確認します。

28.1 完全な積分法

GetFEM++ の正確な積分法のセットを与えます。これは、多項式が正確に積分されていることを意味します。しかし、その使用は（非常に）制限されており、推奨されていません。正確な積分法の使用は、多項式のための低レベル総称構築に限定されています。線形変換を伴う τ 等価要素と線形項では制限されています。高水準の汎用構築でこれらを使用することはできません。

利用可能な正確な積分法のリストは以下の通りです

表 28.1 正確な積分法

"IM_NONE () "	ダミー積分法。
"IM_EXACT_SIMPLEX (n) "	次元 n を参照するシンプレックスの単体における多項式の正確な積分の説明。
"IM_PRODUCT (a, b) "	a と b の凸包の直接の積である凸包面上の正確な積分の説明。
"IM_EXACT_PARALLELEPIPED (n) "	次元 n を参照する平行 6 面体上の多項式の正確な積分の説明。
"IM_EXACT_PRISM (n) "	次元 n を参照するプリズムへの多項式の正確な積分の説明

平行 6 面体またはプリズムに正確な積分法の記述が存在しても、ほとんどの場合、そのような要素の幾何学的変換は非線形であり、正確な積分は使用できません。

28.2 Newton Cotes 積分法

シンプレックス、平行 6 面体、およびプリズムの K 次の Newton Cotes 積分は、"IM_NC(N,K)"、"IM_NC_PARALLELEPIPED(N,K)"、"IM_NC_PRISM(N,K)" とします。

28.3 次元 1 の Gauss 積分法

Gauss-Legendre の K ($K/2+1$ 点) の部分には、"IM_GAUSS1D(K)" と表示されます。Gauss-Lobatto-Legendre 積分は、($K/2+1$ 点を持つ) K 次の部分に、"IM_GAUSSLOBATTO1D(K)" と表示されます。これは K の奇数値に対してのみ利用可能です。Gauss-Lobatto 積分は、集中質量の挙動をするために "FEM_PK_GAUSSLOBATTO1D(K/2)" と一緒に使用して一括処理を実行できます。

28.4 次元 2 の Gauss 積分法

表 28.2 次元 2 の積分法

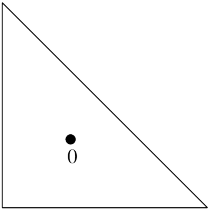
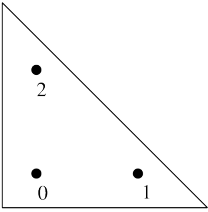
図	座標 (x, y)	重み	呼び出し/次数関数
	(1/3, 1/3)	1/2	"IM_TRIANGLE(1)" 1 点、次数 1。
	(1/6, 1/6) (2/3, 1/6) (1/6, 2/3)	1/6 1/6 1/6	"IM_TRIANGLE(2)" 3 点、次数 2。

表 28.3 次元 2 の積分法





図	座標 (x, y)	重み	呼び出し/次数関数
	(1/3, 1/3)	-27/96	"IM_TRIANGLE (3) "
	(1/5, 1/5)	25/96	4 点、次数 3。
	(3/5, 1/5)	25/96	
	(1/5, 3/5)	25/96	
	(a, a)	c	"IM_TRIANGLE (4) "
	(1-2a, a)	c	6 点、次数 4
	(a, 1-2a)	c	$a = 0.445948490915965$
	(b, b)	d	$b = 0.091576213509771$
	(1-2b, b)	d	$c = 0.111690794839005$
	(b, 1-2b)	d	$d = 0.054975871827661$
	(1/3, 1/3)	9/80	"IM_TRIANGLE (5) "
	(a, a)	c	7 点、次数 5
	(1-2a, a)	c	$a = \frac{6 + \sqrt{15}}{21} \quad b = 4/7 - a \quad c = \frac{155 + \sqrt{15}}{2400}$
	(a, 1-2a)	c	$d = 31/240 - c$
	(b, b)	d	
	(1-2b, b)	d	
	(b, 1-2b)	d	
	(a, a)	e	"IM_TRIANGLE (6) "
	(1-2a, a)	e	12 点、次数 6
	(a, 1-2a)	e	$a = 0.063089104491502$
	(b, b)	f	$b = 0.249286745170910$
	(1-2b, b)	f	$c = 0.310352451033785$
	(b, 1-2b)	f	$d = 0.053145049844816$
	(c, d)	g	$e = 0.025422453185103$
	(d, c)	g	$f = 0.058393137863189$
	(1-c-d, c)	g	$g = 0.041425537809187$
	(1-c-d, d)	g	
	(c, 1-c-d)	g	
(d, 1-c-d)	g		

表 28.4 次元 2 の積分法



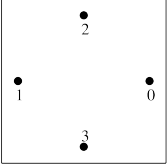
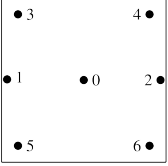
図	座標 (x, y)	重み	呼び出し/次数関数
	(a, a)	h	"IM_TRIANGLE (7) "
	(b, a)	h	13 点、次数 7
	(a, b)	h	$a = 0.0651301029022$ $b = 0.8697397941956$
	(c, e)	i	$c = 0.3128654960049$ $d = 0.6384441885698$
	(d, c)	i	$e = 0.0486903154253$ $f = 0.2603459660790$
	(e, d)	i	$g = 0.4793080678419$ $h = 0.0266736178044$
	(d, e)	i	$i = 0.0385568804451$ $j = 0.0878076287166$
	(c, d)	i	$k = -0.0747850222338$
	(e, c)	i	
	(f, f)	j	
	(g, f)	j	
	(f, g)	j	
	(1/3, 1/3)	k	
	$(1/2 + \sqrt{1/6}, 1/2)$	1/3	"IM_QUAD (2) "
	$((1/2 - \sqrt{1/24}, 1/2 \pm \sqrt{1/8})$	1/3	3 点、次数 2
			"IM_TRIANGLE (8) "
			([EncyclopCubature] を参照してください)
			"IM_TRIANGLE (9) "
		([EncyclopCubature] を参照してください)	
		"IM_TRIANGLE (10) "	
		([EncyclopCubature] を参照してください)	
		“IM_TRIANGLE (13) “	
		([EncyclopCubature] を参照してください)	

表 28.5 次元 2 の積分法

図	座標 (x, y)	重み	呼び出し/次数関数
	$(1/2 \pm \sqrt{1/6}, 1/2)$ $(1/2, 1/2 \pm \sqrt{1/6})$	 1/4 1/4	 "IM_QUAD (3) " 4 点、次数 3
	$(1/2, 1/2)$ $(1/2 \pm \sqrt{7/30}, 1/2)$ $(1/2 \pm \sqrt{1/12}, 1/2 \pm \sqrt{3/20})$	 2/7 5/63 5/36	 "IM_QUAD (5) " 7 点、次数 5 "IM_QUAD (7) " 12 点、次数 7 "IM_QUAD (9) " 20 点、次数 9 "IM_QUAD (17) " 70 点、次数 17

1DGauss の積分の直積である "IM_GAUSS_PARALLELEPIPED (n, k) " もあります。



重要な注記: IM_QUAD (k) は多項式の次数 k に対して厳密であり、 Q_k 多項式の次数 $2 * k$ を持つことを忘れないでください。例えば、IM_QUAD (7) は、2 つの Q_2 多項式の積を正確に積分することはできません。一方、IM_GAUSS_PARALLELEPIPED (2, 4) は、その積を正確に積分することができます...

28.5 次元 3 の Gauss 積分法

表 28.6 次元 3 の積分法

図	座標 (x, y)	重み	呼び出し/次数関数
	(1/4, 1/4, 1/4)	1/6	"IM_TETRAHEDRON (1) " 1 点、次数 1
	(a, a, a) (a, b, a) (a, a, b) (b, a, a)	1/24 1/24 1/24 1/24	"IM_TETRAHEDRON (2) " 4 点、次数 2} hspace{7em} $a = \frac{5 - \sqrt{5}}{20}$ $b = \frac{5 + 3\sqrt{5}}{20}$

表 28.7 次元 3 の積分法

図	座標 (x, y)	重み	呼び出し/次数関数
	(1/4, 1/4, 1/4)	-2/15	"IM_TETRAHEDRON (3) "
	(1/6, 1/6, 1/6)	3/40	5 点、次数 3
	(1/6, 1/2, 1/6)	3/40	
	(1/6, 1/6, 1/2)	3/40	
	(1/2, 1/6, 1/6)	3/40	
	(1/4, 1/4, 1/4)	8/405	"IM_TETRAHEDRON (5) "
	(a, a, a)	<i>h</i>	15 点、次数 5
	(a, a, c)	<i>h</i>	$a = \frac{7 + \sqrt{15}}{34}$
	(a, c, a)	<i>h</i>	$b = \frac{7 - \sqrt{15}}{34}$
	(c, a, a)	<i>h</i>	$c = \frac{13 + 3\sqrt{15}}{34}$
	(b, b, b)	<i>i</i>	$d = \frac{13 - 3\sqrt{15}}{34}$
	(b, b, d)	<i>i</i>	$e = \frac{5 - \sqrt{15}}{20}$
	(b, d, b)	<i>i</i>	$f = \frac{5 + \sqrt{15}}{20}$
	(d, b, b)	<i>i</i>	$h = \frac{2665 - 14\sqrt{15}}{226800}$
	(e, e, f)	5/567	$i = \frac{2665 + 14\sqrt{15}}{226800}$
	(e, f, e)	5/567	
	(f, e, e)	5/567	
	(e, f, f)	5/567	
	(f, e, f)	5/567	
(f, f, e)	5/567		

その他の方法は次のとおりです。

名前	要素型	点数
"IM_TETRAHEDRON (6) "	4 面体	24
"IM_TETRAHEDRON (8) "	4 面体	43
"IM_SIMPLEX4D (3) "	4D シンプレックス	6
"IM_HEXAHEDRON (5) "	3D6 面体	14
"IM_HEXAHEDRON (9) "	3D6 面体	58
"IM_HEXAHEDRON (11) "	3D6 面体	90
"IM_CUBE4D (5) "	4D 平行 6 面体	24
"IM_CUBE4D (9) "	4D 平行 6 面体	145

28.6 積分法の直積

"IM_PRODUCT(IM1, IM2)" を使って、四辺形やプリズムの積分法を作ることができます。これは、2 つの積分法の直積を与えます。例えば、"IM_GAUSS_PARALLELEPIPED(2,k)" は "IM_PRODUCT(IM_GAUSS1D(2,k), IM_GAUSS1D(2,k))" の別名であり、"IM_QUAD" の積分の代わりに使用することができます。

28.7 具体的な積分法

ピラミッド型要素の場合、"IM_PYRAMID(im)" は、6 面体 (例えば "IM_GAUSS_PARALLELEPIPED(3,5)") からピラミッドへ積分 im 変換する積分法を提供します。これは、ピラミッド要素の有理数分数試行関数に特別に適合された特異な積分法です。

28.8 コンポジット積分法

"IM_STRUCTURED_COMPOSITE(IM1, S)" を使って、S の細分を持つ要素に IM1 をコピーしてください。得られた積分法は同じ次数ですが、より多くの点を有します。メソッドの次数を改善するのではなく、複合メソッドを使用の方が安定している可能性があります。それらのメソッドは複合要素でも使用する必要があります。複合要素はほとんどの場合、境界上に点がありません。(勾配がサブ要素の境界上に定義されない可能性があるため) 基本メソッド "IM1" を選択することが望ましいです。

HCT 要素については、複合積分 "IM_HCT_COMPOSITE(im)" (元の 3 角形を 3 つの 3 角形に分割する) を使用することをお勧めします。

ピラミッド型要素の場合、"IM_PYRAMID_COMPOSITE(im)" は、ピラミッドを 2 つの 4 面体に分解する積分法を提供します (im は 4 面体の積分法でなければなりません)。ここで、im が 6 面体の積分法である場合 "IM_PYRAMID(im)" の積分法が望ましいはずです。

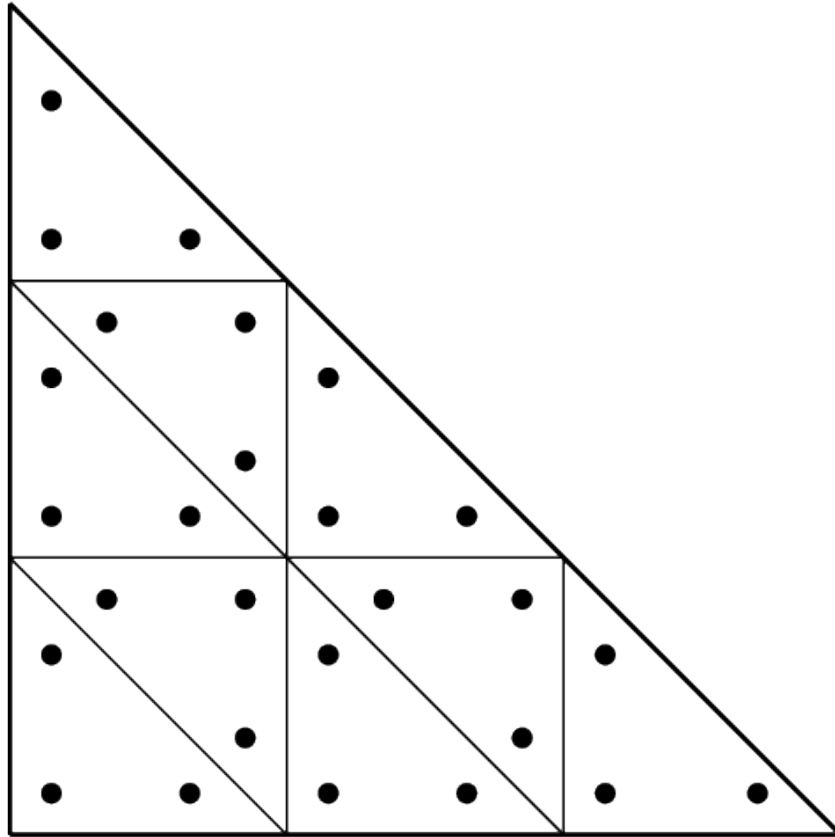


図 28.1 複合メソッド "IM_STRUCTURED_COMPOSITE(IM_TRIANGLE(2), 3)"

第 29 章

References

参考文献

- [AL-CU1991] P. Alart, A. Curnier. *A mixed formulation for frictional contact problems prone to newton like solution methods*. Comput. Methods Appl. Mech. Engrg. 92, 353–375, 1991.
- [Al-Ge1997] E.L. Allgower and K. Georg. *Numerical Path Following*, Handbook of Numerical Analysis, Vol. V (P.G. Ciarlet and J.L. Lions, eds.). Elsevier, pp. 3-207, 1997.
- [AM-MO-RE2014] S. Amdouni, M. Moakher, Y. Renard, *A local projection stabilization of fictitious domain method for elliptic boundary value problems*. Appl. Numer. Math., 76:60-75, 2014.
- [AM-MO-RE2014b] S. Amdouni, M. Moakher, Y. Renard. *A stabilized Lagrange multiplier method for the enriched finite element approximation of Tresca contact problems of cracked elastic bodies*. Comput. Methods Appl. Mech. Engrg., 270:178-200, 2014.
- [bank1983] R.E. Bank, A.H. Sherman, A. Weiser. *Refinement algorithms and data structures for regular local mesh refinement*. In Scientific Computing IMACS, Amsterdam, North-Holland, pp 3-17, 1983.
- [ba-dv1985] K.J. Bathe, E.N. Dvorkin, *A four-node plate bending element based on Mindlin-Reissner plate theory and a mixed interpolation*. Internat. J. Numer. Methods Engrg., 21, 367-383, 1985.
- [Be-Mi-Mo-Bu2005] Bechet E, Minnebo H, Moës N, Burgardt B. *Improved implementation and robustness study of the X-FEM for stress analysis around cracks*. Internat. J. Numer. Methods Engrg., 64, 1033-1056, 2005.
- [BE-CO-DU2010] M. Bergot, G. Cohen, M. Duruflé. *Higher-order finite elements for hybrid meshes using new nodal pyramidal elements* J. Sci. Comput., 42, 345-381, 2010.
- [br-ba-fo1989] F. Brezzi, K.J. Bathe, M. Fortin. *Mixed-interpolated element for Reissner-Mindlin plates*. Internat. J. Numer. Methods Engrg., 28, 1787-1801, 1989.
- [bu-ha2010] E. Burman, P. Hansbo. *Fictitious domain finite element methods using cut elements: I. A stabilized Lagrange multiplier method*. Computer Methods in Applied Mechanics, 199:41-44, 2680-2686, 2010.
- [ca-re-so1994] D. Calvetti, L. Reichel and D.C. Sorensen. *An implicitly restarted Lanczos method for large symmetric eigenvalue problems*. Electronic Transaction on Numerical Analysis}. 2:1-21, 1994.

- [CH-LA-RE2008] E. Chahine, P. Laborde, Y. Renard. *Crack-tip enrichment in the Xfem method using a cut-off function*. Int. J. Numer. Meth. Engng., 75(6):629-646, 2008.
- [CH-LA-RE2011] E. Chahine, P. Laborde, Y. Renard. *A non-conformal eXtended Finite Element approach: Integral matching Xfem*. Applied Numerical Mathematics, 61:322-343, 2011.
- [ciarlet1978] P.G. Ciarlet. *The finite element method for elliptic problems*. Studies in Mathematics and its Applications vol. 4, North-Holland, 1978.
- [ciarlet1988] P.G. Ciarlet. *Mathematical Elasticity*. Volume 1: Three-Dimensional Elasticity. North-Holland, 1988.
- [EncyclopCubature] R. Cools, *An Encyclopedia of Cubature Formulas*, J. Complexity.
- [dh-to1984] G. Dhatt, G. Touzot. *The Finite Element Method Displayed*. J. Wiley & Sons, New York, 1984.
- [Dh-Go-Ku2003] A. Dhooge, W. Govaerts and Y. A. Kuznetsov. *MATCONT: A MATLAB Package for Numerical Bifurcation Analysis of ODEs*. ACM Trans. Math. Software 31, 141-164, 2003.
- [Duan2014] H. Duan. *A finite element method for Reissner-Mindlin plates*. Math. Comp., 83:286, 701-733, 2014.
- [Dr-La-Ek2014] A. Draganis, F. Larsson, A. Ekberg. *Finite element analysis of transient thermomechanical rolling contact using an efficient arbitrary Lagrangian-Eulerian description*. Comput. Mech., 54, 389-405, 2014.
- [Fa-Po-Re2015] M. Fabre, J. Pousin, Y. Renard. *A fictitious domain method for frictionless contact problems in elasticity using Nitsche's method*. preprint, <https://hal.archives-ouvertes.fr/hal-00960996v1>
- [Fa-Pa2003] F. Facchinei and J.-S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems, Vol. II*. Springer Series in Operations Research, Springer, New York, 2003.
- [Georg2001] K. Georg. *Matrix-free numerical continuation and bifurcation*. Numer. Funct. Anal. Optimization 22, 303-320, 2001.
- [GR-GH1999] R.D. Graglia, I.-L. Gheorma. *Higher order interpolatory vector bases on pyramidal elements* IEEE transactions on antennas and propagation, 47:5, 775-782, 1999.
- [GR-ST2015] D. Grandi, U. Stefanelli. *The Souza-Auricchio model for shape-memory alloys* Discrete and Continuous Dynamical Systems, Series S, 8(4):723-747, 2015.
- [HA-WO2009] C. Hager, B.I. Wohlmuth. *Nonlinear complementarity functions for plasticity problems with frictional contact*. Comput. Methods Appl. Mech. Engrg., 198:3411-3427, 2009
- [HA-HA2004] A Hansbo, P Hansbo. *A finite element method for the simulation of strong and weak discontinuities in solid mechanics*. Comput. Methods Appl. Mech. Engrg. 193 (33-35), 3523-3540, 2004.
- [HA-RE2009] J. Haslinger, Y. Renard. *A new fictitious domain approach inspired by the extended finite element method*. Siam J. on Numer. Anal., 47(2):1474-1499, 2009.

-
- [HI-RE2010] Hild P., Renard Y. *Stabilized lagrange multiplier method for the finite element approximation of contact problems in elastostatics*. Numer. Math. 15:1, 101–129, 2010.
- [KH-PO-RE2006] Khenous H., Pommier J., Renard Y. *Hybrid discretization of the Signorini problem with Coulomb friction, theoretical aspects and comparison of some numerical solvers*. Applied Numerical Mathematics, 56/2:163-192, 2006.
- [KI-OD1988] N. Kikuchi, J.T. Oden. *Contact problems in elasticity*. SIAM, 1988.
- [LA-PO-RE-SA2005] Laborde P., Pommier J., Renard Y., Salaun M. *High order extended finite element method for cracked domains*. Int. J. Numer. Meth. Engng., 64:354-381, 2005.
- [LA-RE-SA2010] J. Lasry, Y. Renard, M. Salaun. *eXtended Finite Element Method for thin cracked plates with Kirchhoff-Love theory*. Int. J. Numer. Meth. Engng., 84(9):1115-1138, 2010.
- [KO-RE2014] K. Poullos, Y. Renard, *An unconstrained integral approximation of large sliding frictional contact between deformable solids*. Computers and Structures, 153:75-90, 2015.
- [LA-RE2006] P. Laborde, Y. Renard. *Fixed point strategies for elastostatic frictional contact problems*. Math. Meth. Appl. Sci., 31:415-441, 2008.
- [Li-Re2014] T. Ligurský and Y. Renard. *A Continuation Problem for Computing Solutions of Discretised Evolution Problems with Application to Plane Quasi-Static Contact Problems with Friction*. Comput. Methods Appl. Mech. Engrg. 280, 222-262, 2014.
- [Li-Re2014hal] T. Ligurský and Y. Renard. *Bifurcations in Piecewise-Smooth Steady-State Problems: Abstract Study and Application to Plane Contact Problems with Friction*. Computational Mechanics, 56:1:39-62, 2015.
- [Li-Re2015hal] T. Ligurský and Y. Renard. *A Method of Piecewise-Smooth Numerical Branching*. Z. Angew. Math. Mech., 97:7:815–827, 2017.
- [Mi-Zh2002] P. Ming and Z. Shi, *Optimal L2 error bounds for MITC3 type element*. Numer. Math. 91, 77-91, 2002.
- [Xfem] N. Moës, J. Dolbow and T. Belytschko, *A finite element method for crack growth without remeshing*. Internat. J. Numer. Methods Engrg., 46, 131-150, 1999.
- [Nackenhurst2004] U. Nackenhurst, *The ALE formulation of bodies in rolling contact. Theoretical foundation and finite element approach*. Comput. Methods Appl. Mech. Engrg., 193:4299-4322, 2004.
- [nedelec1991] J.-C. Nedelec. *Notions sur les techniques d'elements finis*. Ellipses, SMAI, Mathematiques & Applications no 7, 1991.
- [NI-RE-CH2011] S. Nicaise, Y. Renard, E. Chahine, *Optimal convergence analysis for the eXtended Finite Element Method*. Int. J. Numer. Meth. Engng., 86:528-548, 2011.
-

- [Pantz2008] O. Pantz *The Modeling of Deformable Bodies with Frictionless (Self-)Contacts*. Archive for Rational Mechanics and Analysis, Volume 188, Issue 2, pp 183-212, 2008.
- [SCHADD] L.F. Pavarino. *Domain decomposition algorithms for the p-version finite element method for elliptic problems*. Luca F. Pavarino. PhD thesis, Courant Institute of Mathematical Sciences}. 1992.
- [PO-NI2016] K. Poulios, C.F. Niordson, *Homogenization of long fiber reinforced composites including fiber bending effects*. Journal of the Mechanics and Physics of Solids, 94, pp 433-452, 2016.
- [remacle2002] J-F. Remacle, M. Shephard, *An algorithm oriented database*. Internat. J. Numer. Methods Engrg., 58, 349-374, 2003.
- [SE-PO-WO2015] A. Seitz, A. Popp, W.A. Wall, *A semi-smooth Newton method for orthotropic plasticity and frictional contact at finite strains*. Comput. Methods Appl. Mech. Engrg. 285:228-254, 2015.
- [SI-HU1998] J.C. Simo, T.J.R. Hughes. *Computational Inelasticity*. Interdisciplinary Applied Mathematics, vol 7, Springer, New York 1998.
- [so-se-do2004] P. Šolín, K. Segeth, I. Doležal, *Higher-Order Finite Element Methods*. Chapman and Hall/CRC, Studies in advanced mathematics, 2004.
- [SO-PE-OW2008] E.A. de Souza Neto, D Perić, D.R.J. Owen. *Computational methods for plasticity*. J. Wiley & Sons, New York, 2008.
- [renard2013] Y. Renard, *Generalized Newton's methods for the approximation and resolution of frictional contact problems in elasticity*. Comput. Methods Appl. Mech. Engrg., 256:38-55, 2013.
- [SU-CH-MO-BE2001] Sukumar N., Chopp D.L., Moës N., Belytschko T. *Modeling holes and inclusions by level sets in the extended finite-element method*. Comput. Methods Appl. Mech. Engrg., 190:46-47, 2001.
- [ZT1989] Zienkiewicz and Taylor. *The finite element method*. 5th edition, volume 3 : Fluids Dynamics.

索引

- add() (組み込み関数), 19
- add_fem_data (C の関数), 105
- add_fem_variable (C の関数), 105
- add_fixed_size_data (C の関数), 105
- add_fixed_size_variable (C の関数), 105
- add_initialized_fem_data (C の関数), 105
- add_initialized_fixed_size_data (C の関数), 105
- add_initialized_scalar_data (C の関数), 105
- add_multiplier (C の関数), 105
- asm, 38
- asm , 66

- complex_rhs (C の関数), 106
- complex_tangent_matrix (C の関数), 106
- complex_variable (C の関数), 106

- fem, 23

- getfem::slicer_apply_deformation (C の関数), 96
- getfem::slicer_boundary (C の関数), 96
- getfem::slicer_build_edges_mesh (C の関数), 97
- getfem::slicer_build_mesh (C の関数), 97
- getfem::slicer_build_stored_mesh_slice (C の関数), 97
- getfem::slicer_complementary (C の関数), 97
- getfem::slicer_cylinder (C の関数), 96
- getfem::slicer_explode (C の関数), 97
- getfem::slicer_half_space (C の関数), 96
- getfem::slicer_intersect (C の関数), 97
- getfem::slicer_isovalues (C の関数), 97
- getfem::slicer_mesh_with_mesh (C の関数), 97
- getfem::slicer_none (C の関数), 96
- getfem::slicer_sphere (C の関数), 96
- getfem::slicer_union (C の関数), 97

- index() (組み込み関数), 20
- is_in() (組み込み関数), 20
- is_only_convexes() (組み込み関数), 20
- is_only_faces() (組み込み関数), 20

- m.is_complex (C の関数), 105
- mesh, 23
- mesh_fem, 23
- mesh_fem_of_variable (C の関数), 106
- mf.basic_dof_on_region() (組み込み関数), 29
- mf.clear() (組み込み関数), 27
- mf.convex_index() (組み込み関数), 27
- mf.dof_on_region() (組み込み関数), 29
- mf.extension_matrix() (組み込み関数), 30
- mf.fem_of_element() (組み込み関数), 27
- mf.first_convex_of_basic_dof() (組み込み関数), 29
- mf.get_qdim() (組み込み関数), 29
- mf.ind_basic_dof_of_element() (組み込み関数), 28
- mf.is_reduced() (組み込み関数), 30
- mf.linked_mesh() (組み込み関数), 27
- mf.nb_basic_dof() (組み込み関数), 29
- mf.nb_basic_dof_of_element() (組み込み関数), 28
- mf.nb_dof() (組み込み関数), 30
- mf.point_of_basic_dof() (組み込み関数), 29
- mf.reduce_to_basic_dof() (組み込み関数), 30
- mf.reduction_matrix() (組み込み関数), 30
- mf.reference_point_of_basic_dof() (組み込み関数), 29
- mf.set_reduction() (組み込み関数), 30
- mf.set_reduction_matrices() (組み込み関数), 30
- mim.clear() (組み込み関数), 35
- mim.convex_index() (組み込み関数), 35
- mim.int_method_of_element() (組み込み関数), 35
- mim.linked_mesh() (組み込み関数), 35
- mymesh.clear() (組み込み関数), 19
- mymesh.convex_area_estimate() (組み込み関数), 19
- mymesh.convex_index() (組み込み関数), 17
- mymesh.convex_quality_estimate() (組み込み関数), 19
- mymesh.convex_radius_estimate() (組み込み関数), 19
- mymesh.convex_to_point() (組み込み関数), 18
- mymesh.dim() (組み込み関数), 17
- mymesh.has_region() (組み込み関数), 19
- mymesh.ind_points_of_convex() (組み込み関数), 18
- mymesh.is_convex_having_neighbour() (組み込み関数), 19
- mymesh.neighbour_of_convex() (組み込み関数), 18
- mymesh.neighbours_of_convex() (組み込み関数), 18
- mymesh.normal_of_face_of_convex() (組み込み関数), 19
- mymesh.optimize_structure() (組み込み関数), 19
- mymesh.points_index() (組み込み関数), 17
- mymesh.points_of_convex() (組み込み関数), 18
- mymesh.read_from_file() (組み込み関数), 22
- mymesh.region() (組み込み関数), 19
- mymesh.structure_of_convex() (組み込み関数), 18
- mymesh.trans_of_convex() (組み込み関数), 19
- mymesh.write_to_file() (組み込み関数), 21

- Nitsche の方法, 122

- real_rhs (C の関数), 106
- real_tangent_matrix (C の関数), 106
- real_variable (C の関数), 106

- sup() (組み込み関数), 19

- モデル, 102, 103, 113–115, 118–120, 122, 126–128, 130–132, 134, 142, 156, 164, 172, 182, 199
- モデルブリック, 113–115, 119, 132, 134, 142, 164, 172, 182, 199
- モデル項, 103
- モデル要素, 102, 118, 120, 122, 126–128, 130, 131, 156
- 汎用構築, 38, 66