Tutorial

リリース *5.3*

Yves Renard, Julien Pommier, Konstantinos Poulios

2019 年 03 月 26 日

Contents

Chapter 1

前書き

このチュートリアルでは、コメント付きの簡単な例でライブラリ *GetFEM++* の主な特徴を示すことを目的としています。 *GetFEM++* は連成を含む複雑な問題の数値モデリングが可能です。弱形式記述言語のおかげで、任意の連成項を追加するのは比較的簡単です。弱形式のフレーズはアセンブリ中に実行される最適化された命令リストの形式でコンパイルされます。

提供されている主な機能は次のとおりです

- マルチフィジックス問題の近似と汎用アセンブリ。
- 固定サイズ変数。
- 接触問題。
- Fictitious 領域の機能と変換による、メッシュ間または領域間のアセンブリ。
- 継続/分岐問題。

当然、このチュートリアル以外にもいくつかのドキュメンテーションが存在します。実装されたメソッドの説明に関しては、(C++ ライブラリの) ユーザードキュメント を参考にしてください。インターフェイスのドキュメンテーションはインターフェイスのさまざまな機能の使用方法を記載していますが、メソッドの説明は繰り返しません。「開発者の手引き」には、ライブラリの構造に関する内部情報が記載されています。

GetFEM++ は Savannah サイトによってホストされているフリーな共同プロジェクトです (https://savannah.nongnu.org/projects/getfem を参照してください)。新しいコントリビューターは、プロジェクトのあらゆる面で歓迎いたします。

1.1 C++、Python、Scilab、または Matlab?

GetFEM++ は基本的に C++ で書かれたライブラリです。しかし、Python、Scilab や Matlab のインターフェイスでも利用可能であり、すべての高度な機能を使用することができます。プログラミングがより簡単なスクリプト言語のインターフェイスから入門することをお勧めします。3 種類のインターフェイスは、グラフィックの後処理を除けば、異なるのはわずかな構文のみです (Python インターフェイスまたは C++ で直接ライブラリを使用する場合、Paraview Mayavi または gmsh などの専用ソフトウェアを使用しますが、Scilab および Matlab インターフェイスを使用する場合は統合された後処理機能が使用可能です)。デフォルトでは Python インターフェイスが C++ ライブラリと一緒にコンパイルされます。

GetFEM++ のインターフェイスを使用することは優れた戦略です。複雑なアプリケーションの場合でも、パフォーマンスは C++ ライブラリを直接使用した場合に匹敵し、より柔軟な使用が可能です。しかし、(MPI で) 並列化が可能なのは Python インタフェースだけです。インターフェイスへの機能の追加は比較的簡単な作業です。

チュートリアルの最初の (熱電気弾性連成の) 例では、C++ ライブラリとインターフェイスの使用方法の違いを確認することができます。

1.2 デモファイルはどこにありますか?

多数のデモンストレーションプログラムが以下にあります。

- C++ の例は以下のディレクトリ内にあります。

  ```
  tests/
  ```

- Python インターフェイスは以下のディレクトリ内にあります。

  ```
  interface/tests/python
  ```

- Scilab インターフェイスは以下のディレクトリ内にあります。

  ```
  interface/scr/scilab/demos
  ```

- そして、 Matlab インターフェイスは以下のディレクトリ内にあります。

  ```
  interface/tests/matlab
  ```

Chapter 2

インストール方法

GetFEM++ は linux (ubuntu) 上で開発しされているため linux 特に Debian ベースのディストリビューション (Debian/Ubuntu/Mint) でのインストールが簡単です。しかし、Mac os X および Windows 上の他の linux ディストリビューションにも *GetFEM++* をインストールすることはできます。ソースから *GetFEM++* をコンパイルするには、(C++ 11 標準規格をサポートした) 最近の C++ コンパイラと最新バージョンの python 2 が必要です。

GetFEM++ の他のライブラリとの主な依存関係は次のとおりです。

- 最新の変更を取得するために git バージョンからバイナリをビルドする場合は、git クライアント、automake、autoconf、および libtool が必要です。
- python インターフェイスを構築する場合は、 (Python.h などの) python 開発ファイルおよび *numpy* と *scipy* パッケージが必要です。
- *GetFEM++* と一緒に配布されされている SuperLU の代わりに、(疎行列の直接法ソルバーの) 直列 MUMPS パッケージを使用することも可能です。
- MPI で並列化された *GetFEM++* を使用したい場合、 METIS と MPI4PY の並列版のパッケージが必要です。
- メッシュ生成と Fictitious Domain 法のための qhull パッケージが必要です。
- BLAS と LAPACK パッケージが必要です。

GetFEM++ の C++ ライブラリは単独でビルドすることも、Python、Scilab、および/または Matlab インターフェイスと一緒にビルドすることも可能です。

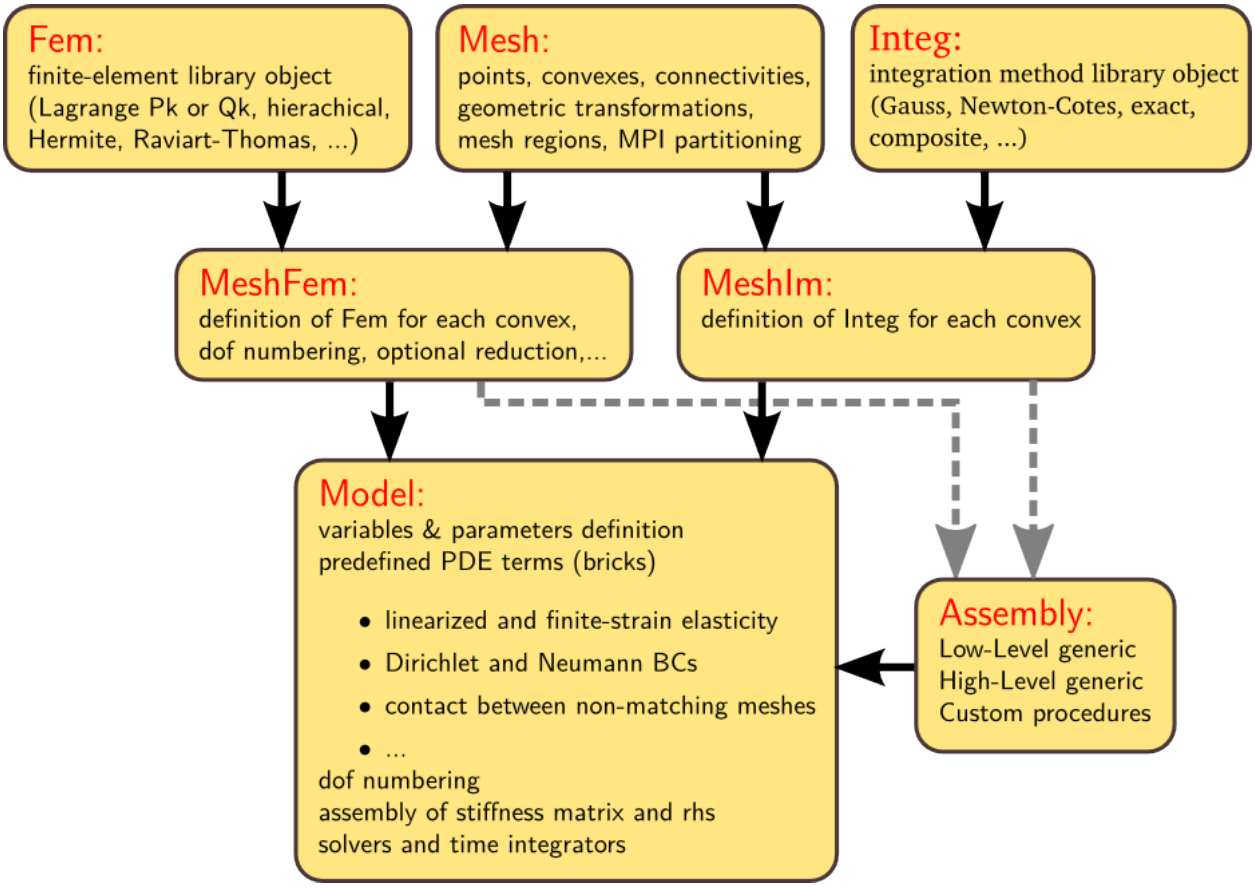
また、対応するパッケージ管理システムを使用して linux ディストリビューションに Getfem の安定版をインストールすることもできます。

Getfem の C++ ライブラリをビルドする方法の詳細については、 ダウンロードしてインストールする を参照してください。

Chapter 3

GetFEM++ の基本的な使い方

GetFEM++ の構造は以下のようにまとめることができます



getfem を使用するプログラムは、多くの場合、次の構造となります

```
... define one or more Mesh

... define one or more MeshFem
```

```
... define one or more MeshIm

... define a Model and set it up:

    Model.add_fem_variable(MeshFem, "some variable name")

    Model.add_fem_variable(MeshFem, "another variable name")

    Model.add_fem_data(MeshFem, "some data name")

    Model.add_nonlinear_generic_assembly_brick(MeshIm,
                            "copy & paste your PDE weak formulation here", MeshRegion)

    Model.solve(...options)
```

弱形式言語で偏微分方程式項を定義するのではなく (弱形式言語の記号の詳細については *ud-gasm-high* を参照してください)、標準項として定義済みのブリックを使用することもできます。汎用的な楕円項、線形化または有限ひずみ弾性、標準境界条件... など。

Chapter 4

熱および電気弾性連成の例 (単純な非線形結合問題、モデルオブジェクト、汎用アセンブリ、解析および可視化)

この例では、変位フィールド、温度フィールド、および電位フィールドの非線形連成のマルチフィジックス問題の簡単な例を紹介します。また、C++ ライブラリと各インターフェイスの使用法を比較することも目的としています。対応するデモファイルは、 *GetFEM++* のテストディレクトリにあります (*tests/* 、 *interface/tests/python* 、 *interface/scr/scilab/demos* 、 *interface/tests/matlab*)。

4.1 問題の設定

$\Omega \subset \mathbb{R}^2$ は、2 次元板の参照配置です (**ここ** のジオメトリを参照してください)、ここで厚さ ε は外部力、電位および加熱に寄与します。ここで、 $\theta : \Omega \to \mathbb{R}$ は (°C 内の) 温度フィールド、 $V : \Omega \to \mathbb{R}$ は電位フィールドです。また、 $u : \Omega \to \mathbb{R}^2$ は膜変位場です。

4.1.1 熱問題

プレートの表裏面は、熱伝達係数 D の (20 °C の) 空気と熱交換しており、プレートの側面は断熱であると想定されています。

熱方程式 θ と境界条件は次のように書くことができます。

$$\begin{cases} -\text{div}(\varepsilon\kappa(\nabla\theta)) + 2D(\theta - T_0) - \varepsilon\sigma|\nabla V|^2 = 0 \quad \text{in } \Omega, \\ \kappa\nabla\theta \cdot n = 0 \quad \text{on } \partial\Omega, \end{cases}$$

κ は熱電導率、 T_0 は空気の温度、 $\partial\Omega$ は、領域 Ω の境界であり n は $\partial\Omega$ 上の Ω への外向きの単位法線ベクトルです。

項 $\sigma|\nabla V|^2$ は、Joule 加熱項に対応する非線形連成項であり、 σ は電気伝導率です。

4.1.2 電位問題

プレートの左右の側面の間には $0.1V$ の電位差があるとします。他の面は電気的に絶縁されていると考えます。電位の方程式は次の通りです。

$$\begin{cases} -\text{div}(\varepsilon\sigma(\nabla V)) = 0 \quad \text{in } \Omega, \\ \sigma\nabla V \cdot n = 0 \quad \text{on the top an bottom lateral faces}, \\ V = 0 \quad \text{on the right lateral face}, \\ V = 0.1 \quad \text{on the left lateral face}, \end{cases}$$

ここでも σ は電気伝導率です。さらに、 σ は次のように温度に依存すると考えます。

$$\sigma = \frac{1}{\rho_0(1 + \alpha(\theta - T_0))},$$

ここで T_0 は参照温度 (ここでは空気温度) であり、 ρ_0 は T_0 上の抵抗温度係数であり、α は第 2 抵抗温度係数です。

4.1.3 変形問題

右側面に加わる力の下でプレートの微小変形を考慮し, プレートの加熱によって影響を受けるとします。変位 u は、次の (線形化された弾性) 問題の解になります。

$$\begin{cases} -\text{div}(\bar{\sigma}(u)) = 0 \quad \text{in } \Omega, \\ \bar{\sigma}\, n = 0 \quad \text{on the top an bottom lateral faces}, \\ \bar{\sigma}\, n = F \quad \text{on the right lateral face}, \\ u = 0 \quad \text{on the left lateral face}, \end{cases}$$

ここで F は右横方向の境界に加えられる力密度であり、 $\bar{\sigma}(u)$ は次のように定義された Cauchy 応力テンソルです。

$$\bar{\sigma}(u) = \lambda^* \text{div}(u) I + 2\mu\bar{\varepsilon}(u) + \beta(T_0 - \theta)I,$$

$\bar{\varepsilon}(u) = (\nabla u + (\nabla u)^T)/2$ は、線形化されたひずみテンソルであり、 I は 2 次単位テンソルで、 λ^*, μ は Lamé 係数であり次のように定義されます。

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)},$$
$$\mu = \frac{E}{2(1+\nu)},$$
$$\lambda^* = \frac{2\lambda\mu}{\lambda + 2 * \mu},$$

E は材料のヤング係数であり ν はポアソン比です。

項 $\beta(T_0 - \theta)I$ は、熱膨張の項に対応しています。ここで $\beta = \alpha_{th}E/(1-2\nu)$ で、 α_{th} は熱膨張係数です。

4.2 弱定式化

方程式の連成系方程式の弱定式化は重要なステップです。有限要素の定式化は弱定式化 (Galerkin 近似) に基づいており、弱定式化は追加される項を表現する唯一の方法であるため、これはきわめて重要なステップです。

各偏微分方程式の弱定式化は、未知数に対応する試行関数を式に掛けることにより得られます。未知数は Dirichlet 条件を満たし条件は均質であるとします。その後 Ω 領域上の積分と (Green の式を使用した) 部分のいくつかの積分を実行します。偏微分方程式系の弱定式化は次の通りです。

$$\text{Find } \theta, V, u \text{ with } V = 0.1, u = 0 \text{ on the left face}, V = 0 \text{ on the right face},$$
$$\int_\Omega \varepsilon\kappa\nabla\theta \cdot \nabla\delta_\theta + 2D\theta\delta_\theta dx = \int_\Omega (2DT_0 + \varepsilon\sigma|\nabla V|^2)\delta_\theta dx \quad \text{for all } \delta_\theta,$$
$$\int_\Omega \varepsilon\sigma\nabla V \cdot \nabla\delta_V = 0 dx \quad \text{for all } \delta_V \text{ satisfying } \delta_V = 0 \text{ on the left and right faces},$$
$$\int_\Omega \bar{\sigma}(u) : \bar{\varepsilon}(\delta_u) dx = \int_{\Gamma_N} F \cdot \delta_u d\Gamma \quad \text{for all } \delta_u \text{ satisfying } \delta_u = 0 \text{ on the left face},$$

ここで $\delta_\theta, \delta_V, \delta_u$ は、それぞれ、 θ, V, u に対応する試行関数です、 Γ_N は力密度 F がかかる右の境界を示します。そして $\bar{\sigma} : \bar{\varepsilon}$ は 2 次テンソル同士の Frobenius スカラー積です。

4.3 C++ での実装とインターフェイス

次に、問題を近似する *GetFEM++* の使用法についての詳細なプレゼンテーションを行います。C++、Python、Scilab と Matlab プログラムを同時に構築します。Matlab および Scilab プログラムでは、オブジェクト指向コマンドを使用しません (使用する方法は *mlab-oocmd* を参照してください)

4.3.1 初期化

まず、C++ では、モデルオブジェクト、ジェネリックアセンブリ、線形代数インターフェイス (Gmm++) 、実験用メッシュ、およびエクスポート機能に対して、特定の数のヘッダーをインクルードしなければなりません。Python の場合、これは簡単です。 *GetFEM++* をグローバルにインポートするだけです (numpy もインポートする必要があります)。Scilab の場合、最初にライブラリを Scilab コンソールにロードします (ここでは説明しません)、Matlab の場合、すべての *GetFEM++* 変数をクリアする *gf_workspace('clear all')* 以外、何も必要ありません。

C++

```
#include "getfem/getfem_model_solvers.h"
#include "getfem/getfem_export.h"
#include "gmm/gmm.h"
#include "getfem/getfem_mesher.h"
#include "getfem/getfem_generic_assembly.h"

using bgeot::size_type;
using bgeot::base_node;
using bgeot::base_small_vector;
typedef getfem::model_real_plain_vector plain_vector;

int main(void) {
```

Python

```
import getfem as gf
import numpy as np
```

Scilab

```
gf_workspace('clear all'); // In case of multiple runs
```

Matlab

```
gf_workspace('clear all'); % In case of multiple runs
```

4.3.2 モデルのパラメータ

ここで、問題のさまざまな物理パラメータおよび数値パラメータを定義しましょう。スクリプト言語 (Python、Scilab と Matlab) には違いはありません。

C++

```
double epsilon = 1.; // Thickness of the plate (cm)
double E = 21E6;     // Young Modulus (N/cm^2)
double nu = 0.3;     // Poisson ratio
double clambda = E*nu/((1+nu)*(1-2*nu));
double cmu = E/(2*(1+nu));
double clambdastar = 2*clambda*cmu/(clambda+2*cmu);
double F = 100E2;    // Force density at the right boundary (N/cm^2)
double kappa = 4.;   // Thermal conductivity (W/(cm K))
double D = 10.;      // Heat transfer coefficient (W/(K cm^2))
double air_temp = 20;// Temperature of the air in oC.
double alpha_th = 16.6E-6; // Thermal expansion coefficient (/K)
double T0 = 20.;     // Reference temperature in oC
double rho_0 = 1.754E-8; // Resistance temperature coefficient at T0
double alpha = 0.0039; // Second resistance temperature coefficient

double h = 2.        // Approximate mesh size
bgeot::dim_type elements_degree = 2; // Degree of the finite element
 ↪  methods
```

Scripts

```
epsilon = 1.; E = 21E6; nu = 0.3;
clambda = E*nu/((1+nu)*(1-2*nu));
cmu = E/(2*(1+nu));
clambdastar = 2*clambda*cmu/(clambda+2*cmu);
F = 100E2; kappa = 4.; D = 10;
air_temp = 20; alpha_th = 16.6E-6;
T0 = 20; rho_0 = 1.754E-8;
alpha = 0.0039;

h = 2; elements_degree = 2;
```

4.3.3 メッシュ生成

GetFEM++ には、ここで説明するいくつかの制約のあるメッシュ機能があります。ここではそれらを使用するつもりです。しかし、得られたメッシュの品質と適合性は保証していません。そのため、 *GetFEM++* のメッシュ作成機能を使用する場合はメッシュを確認することをお勧めします。また、外部メッシャ (例えば GiD または Gmsh) を使用してインポートすることもできます (*ud-load_save_mesh* をご覧ください)。

領域のジオメトリは、3 つの円形の穴がある長方形であるとします (**得られたメッシュ。** を参照)。ジオメトリは、いくつかの初等幾何と union/setminus 操作によって記述します (`src/getfem/getfem_mesher.h` ファイルを参照してください)。以下では、 *h* はメッシュサイズを表し、 *2* はメッシュの次数を表します (これは変換が次数 2 であり、曲線エッジを使用することを意味します)。

C++

```
getfem::mesh mesh;
getfem::pmesher_signed_distance
mo1 = getfem::new_mesher_rectangle(base_node(0., 0.), base_node(100.,
↪  25.)),
mo2 = getfem::new_mesher_ball(base_node(25., 12.5), 8.),
mo3 = getfem::new_mesher_ball(base_node(50., 12.5), 8.),
mo4 = getfem::new_mesher_ball(base_node(75., 12.5), 8.),
mo5 = getfem::new_mesher_union(mo2, mo3, mo4),
mo = getfem::new_mesher_setminus(mo1, mo5);

std::vector<getfem::base_node> fixed;
getfem::build_mesh(mesh, mo, h, fixed, 2, -2);
```

Python

```
mo1 = gf.MesherObject('rectangle', [0., 0.], [100., 25.])
mo2 = gf.MesherObject('ball', [25., 12.5], 8.)
mo3 = gf.MesherObject('ball', [50., 12.5], 8.)
mo4 = gf.MesherObject('ball', [75., 12.5], 8.)
mo5 = gf.MesherObject('union', mo2, mo3, mo4)
mo  = gf.MesherObject('set minus', mo1, mo5)

mesh = gf.Mesh('generate', mo, h, 2)
```

Scilab

```
mo1 = gf_mesher_object('rectangle', [0 0], [100 25]);
mo2 = gf_mesher_object('ball', [25 12.5], 8);
mo3 = gf_mesher_object('ball', [50 12.5], 8);
mo4 = gf_mesher_object('ball', [75 12.5], 8);
mo5 = gf_mesher_object('union', mo2, mo3, mo4);
mo  = gf_mesher_object('set minus', mo1, mo5);

mesh = gf_mesh('generate', mo, h, 2);
```

Matlab

```
mo1 = gf_mesher_object('rectangle', [0 0], [100 25]);
mo2 = gf_mesher_object('ball', [25 12.5], 8);
mo3 = gf_mesher_object('ball', [50 12.5], 8);
mo4 = gf_mesher_object('ball', [75 12.5], 8);
mo5 = gf_mesher_object('union', mo2, mo3, mo4);
mo  = gf_mesher_object('set minus', mo1, mo5);

mesh = gf_mesh('generate', mo, h, 2);
```

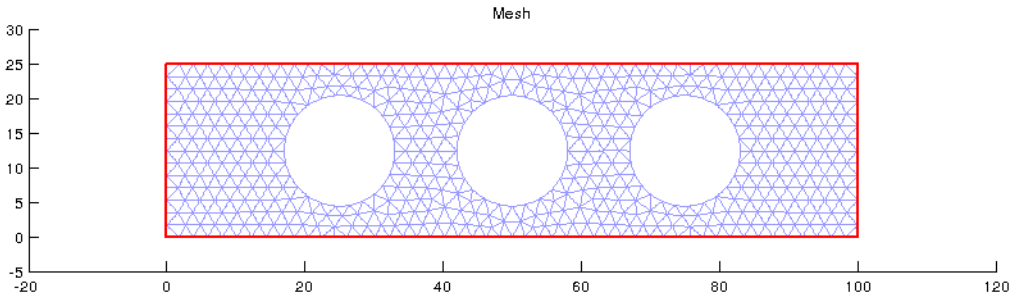


Figure 4.1: 得られたメッシュ。

4.3.4 境界の選択

境界のそれぞれの部分には異なる境界条件を設定するため、境界のさまざまな部分には番号を付けます (穴には、熱と電気の絶縁と、応力自由境界条件が想定されます)。したがって、メッシュ上の要素面を選択し、メッシュ領域を定義する必要があります (*ud-mesh_regions* を参照)、1、2、3、4 はそれぞれ右境界、左境界、上境界、下境界です。これらの境界番号は、モデルのブリックで使用されます。

C++

```
getfem::mesh_region border_faces;
getfem::outer_faces_of_mesh(mesh, border_faces);
getfem::mesh_region fb1
  = getfem::select_faces_in_box(mesh, border_faces, base_node(1., 1.),
                 base_node(99., 24.));
getfem::mesh_region fb2
  = getfem::select_faces_of_normal(mesh, border_faces,
                     base_small_vector( 1., 0.), 0.01);
getfem::mesh_region fb3
  = getfem::select_faces_of_normal(mesh, border_faces,
                     base_small_vector(-1., 0.), 0.01);
getfem::mesh_region fb4
  = getfem::select_faces_of_normal(mesh, border_faces,
                     base_small_vector(0.,  1.), 0.01);
getfem::mesh_region fb5
  = getfem::select_faces_of_normal(mesh, border_faces,
                     base_small_vector(0., -1.), 0.01);

size_type RIGHT_BOUND=1, LEFT_BOUND=2, TOP_BOUND=3, BOTTOM_BOUND=4;
mesh.region( RIGHT_BOUND) = getfem::mesh_region::subtract(fb2, fb1);
mesh.region(  LEFT_BOUND) = getfem::mesh_region::subtract(fb3, fb1);
mesh.region(   TOP_BOUND) = getfem::mesh_region::subtract(fb4, fb1);
mesh.region(BOTTOM_BOUND) = getfem::mesh_region::subtract(fb5, fb1);
```

Python

```
fb1 = mesh.outer_faces_in_box([1., 1.], [99., 24.])
fb2 = mesh.outer_faces_with_direction([ 1., 0.], 0.01)
fb3 = mesh.outer_faces_with_direction([-1., 0.], 0.01)
fb4 = mesh.outer_faces_with_direction([0.,  1.], 0.01)
fb5 = mesh.outer_faces_with_direction([0., -1.], 0.01)

RIGHT_BOUND=1; LEFT_BOUND=2; TOP_BOUND=3; BOTTOM_BOUND=4;
 ↪  HOLE_BOUND=5;

mesh.set_region( RIGHT_BOUND, fb2)
mesh.set_region(  LEFT_BOUND, fb3)
mesh.set_region(   TOP_BOUND, fb4)
mesh.set_region(BOTTOM_BOUND, fb5)
mesh.set_region(  HOLE_BOUND, fb1)
mesh.region_subtract( RIGHT_BOUND, HOLE_BOUND)
mesh.region_subtract(  LEFT_BOUND, HOLE_BOUND)
mesh.region_subtract(   TOP_BOUND, HOLE_BOUND)
mesh.region_subtract(BOTTOM_BOUND, HOLE_BOUND)
```

Scilab

```
fb1 = gf_mesh_get(mesh, 'outer faces in box', [1 1], [99 24]);
fb2 = gf_mesh_get(mesh, 'outer faces with direction', [ 1 0], 0.01);
fb3 = gf_mesh_get(mesh, 'outer faces with direction', [-1 0], 0.01);
fb4 = gf_mesh_get(mesh, 'outer faces with direction', [0  1], 0.01);
fb5 = gf_mesh_get(mesh, 'outer faces with direction', [0 -1], 0.01);

RIGHT_BOUND=1; LEFT_BOUND=2; TOP_BOUND=3; BOTTOM_BOUND=4;
 ↪  HOLE_BOUND=5;
gf_mesh_set(mesh, 'region',  RIGHT_BOUND, fb2);
gf_mesh_set(mesh, 'region',   LEFT_BOUND, fb3);
gf_mesh_set(mesh, 'region',    TOP_BOUND, fb4);
gf_mesh_set(mesh, 'region', BOTTOM_BOUND, fb5);
gf_mesh_set(mesh, 'region',   HOLE_BOUND, fb1);
gf_mesh_set(mesh, 'region subtract',  RIGHT_BOUND, HOLE_BOUND);
gf_mesh_set(mesh, 'region subtract',   LEFT_BOUND, HOLE_BOUND);
gf_mesh_set(mesh, 'region subtract',    TOP_BOUND, HOLE_BOUND);
gf_mesh_set(mesh, 'region subtract', BOTTOM_BOUND, HOLE_BOUND);
```

Matlab

```
fb1 = gf_mesh_get(mesh, 'outer faces in box', [1 1], [99 24]);
fb2 = gf_mesh_get(mesh, 'outer faces with direction', [ 1 0], 0.01);
fb3 = gf_mesh_get(mesh, 'outer faces with direction', [-1 0], 0.01);
fb4 = gf_mesh_get(mesh, 'outer faces with direction', [0  1], 0.01);
fb5 = gf_mesh_get(mesh, 'outer faces with direction', [0 -1], 0.01);

RIGHT_BOUND=1; LEFT_BOUND=2; TOP_BOUND=3; BOTTOM_BOUND=4;
 ↪  HOLE_BOUND=5;
gf_mesh_set(mesh, 'region',  RIGHT_BOUND, fb2);
gf_mesh_set(mesh, 'region',   LEFT_BOUND, fb3);
gf_mesh_set(mesh, 'region',    TOP_BOUND, fb4);
gf_mesh_set(mesh, 'region', BOTTOM_BOUND, fb5);
gf_mesh_set(mesh, 'region',   HOLE_BOUND, fb1);
gf_mesh_set(mesh, 'region subtract',  RIGHT_BOUND, HOLE_BOUND);
gf_mesh_set(mesh, 'region subtract',   LEFT_BOUND, HOLE_BOUND);
gf_mesh_set(mesh, 'region subtract',    TOP_BOUND, HOLE_BOUND);
gf_mesh_set(mesh, 'region subtract', BOTTOM_BOUND, HOLE_BOUND);
```

4.3.5 メッシュの描画

メッシュをプレビューし、その妥当性を制御するために、次の手順を使用します。

C++

```
getfem::vtk_export exp("mesh.vtk", false);
exp.exporting(mesh);
exp.write_mesh();
// You can view the mesh for instance with
// mayavi2 -d mesh.vtk -f ExtractEdges -m Surface
```

Python

```
mesh.export_to_vtk('mesh.vtk');
# You can view the mesh for instance with
# mayavi2 -d mesh.vtk -f ExtractEdges -m Surface
```

Scilab

```
scf(1);
gf_plot_mesh(mesh, 'refine', 8, 'curved', 'on', 'regions', ...
             [RIGHT_BOUND LEFT_BOUND TOP_BOUND BOTTOM_BOUND]);
title('Mesh');
sleep(1000);
```

Matlab

```
gf_plot_mesh(mesh, 'refine', 8, 'curved', 'on', 'regions', ...
             [RIGHT_BOUND LEFT_BOUND TOP_BOUND BOTTOM_BOUND]);
title('Mesh');
pause(1);
```

C++ および Python インターフェイスでは、外部グラフィカルポストプロセッサを使用する必要があります (たとえば、gmsh、Mayavi2、または Paraview)。Scilab および Matlab インタフェースでは、内部プロット機能を使用することができます (結果は **得られたメッシュ。** を参照)。

4.3.6 有限要素法と積分法の定義

3 つの有限要素法を定義します。最初の 1 つは、変位フィールドを近似する *mfu* です。これはベクトルフィールドで C++ では次のように定義されます。

```
getfem::mesh_fem mfu(mesh, 2);
mfu.set_classical_finite_element(elements_degree);
```

ここで、*2* はベクトル場の次元を表します。2 行目は、使用する有限要素を設定します。*classical_finite_element* は、連続した Lagrange 要素を意味し、*elements_degree* は *2* に設定されています。これは 2 次の (アイソパラメトリック) 要素を使用することを意味します。

GetFEM++ では、既存の有限要素法を幅広く選択肢できます。 *ud-appendixa* を参照してください。しかし、実際には Lagrange 有限要素法が最も使用されています。

第 2 の有限要素法はスカラーで、温度場と電位場の両方を近似する *mft* です。単一の有限要素法で任意の数の有限要素変数を近似すると便利です。

3 番目の有限要素法は不連続なスカラー Lagrange で、1 変数の導関数を補間することができます (たとえば、Von Mises 応力の補間)。

最後に定義するのは、積分法 *mim* です。*GetFEM++* にデフォルトの積分法はありません。したがって、これは積分法を定義するためには必須です。もちろん、積分法の次数は、選択された有限要素法に好都合な積分を行うため、十分に選定しなければなりません。ここでは、 *elements_degree* の 2 乗で十分です。

C++

```
getfem::mesh_fem mfu(mesh, 2);
mfu.set_classical_finite_element(elements_degree);
getfem::mesh_fem mft(mesh, 1);
mft.set_classical_finite_element(elements_degree);
getfem::mesh_fem mfvm(mesh, 1);
mfvm.set_classical_discontinuous_finite_element(elements_degree);

getfem::mesh_im  mim(mesh);
mim.set_integration_method(bgeot::dim_type(gmm::sqr(elements_degree)));
```

Python

```
mfu = gf.MeshFem(mesh, 2)
mfu.set_classical_fem(elements_degree)
mft = gf.MeshFem(mesh, 1)
mft.set_classical_fem(elements_degree)
mfvm = gf.MeshFem(mesh, 1)
mfvm.set_classical_discontinuous_fem(elements_degree)
mim = gf.MeshIm(mesh, pow(elements_degree,2))
```

Scilab

```
mfu = gf_mesh_fem(mesh, 2);
gf_mesh_fem_set(mfu, 'classical fem', elements_degree);
mft = gf_mesh_fem(mesh, 1);
gf_mesh_fem_set(mft, 'classical fem', elements_degree);
mfvm = gf_mesh_fem(mesh, 1);
gf_mesh_fem_set(mfvm, 'classical discontinuous fem',
 ↪  elements_degree-1);
mim = gf_mesh_im(mesh, elements_degree^2);
```

Matlab

```
mfu = gf_mesh_fem(mesh, 2);
gf_mesh_fem_set(mfu, 'classical fem', elements_degree);
mft = gf_mesh_fem(mesh, 1);
gf_mesh_fem_set(mft, 'classical fem', elements_degree);
mfvm = gf_mesh_fem(mesh, 1);
gf_mesh_fem_set(mfvm, 'classical discontinuous fem',
 ↪  elements_degree-1);
mim = gf_mesh_im(mesh, elements_degree^2);
```

4.3.7 モデルの定義

GetFEM++ のモデルオブジェクトは (未知) 変数、データ、およびモデルブリックと呼ばれるものの集まりです。モデルブリックは、単一の変数か、複数の変数をリンクしているモデルの一部 (線形または非線形項) です。これらは、(接線) 線形システムの構築のために使用されます (詳細については *ud-model-object* を参照してください)。

直接構築を行い (接線) 線形システムを作成することも可能なため、モデルオブジェクトを使用することは厳密には必須ではありません。しかし、モデルオブジェクトは、モデルのほとんどの既往な部分が事前にプログラムされているので、モデルの素早い構築が可能です。 標準境界条件、標準偏微分方程式、制約を処理するための乗数の使用などが整備されています。さらに、いくつかのブリックは、標準ブリック (ジェネリックアセンブリブリック、陽なマトリックスブリックなど) の拡張が可能なように設計されています。そのため、モデルオブジェクトのフレームワークを使用することをお勧めします。

モデルには実数と複素数の 2 つのバージョンがあります。複素数モデルは複雑な線形システムを解くのに有利な特殊なアプリケーション (例えば、いくつかの電磁気学の問題など) のために予約されています。

計算される 3 つのフィールドに対応する 3 つの変数を持つ実際のモデルを宣言してみましょう 。

C++

```
getfem::model md;
md.add_fem_variable("u", mfu);
md.add_fem_variable("theta", mft);
md.add_fem_variable("V", mft);
```

Python

```
md=gf.Model('real');
md.add_fem_variable('u', mfu)
md.add_fem_variable('theta', mft)
md.add_fem_variable('V', mft)
```

Scilab

```
md=gf_model('real');
gf_model_set(md, 'add fem variable', 'u', mfu);
gf_model_set(md, 'add fem variable', 'theta', mft);
gf_model_set(md, 'add fem variable', 'V', mft);
```

Matlab

```
md=gf_model('real');
gf_model_set(md, 'add fem variable', 'u', mfu);
gf_model_set(md, 'add fem variable', 'theta', mft);
gf_model_set(md, 'add fem variable', 'V', mft);
```

4.3.8 弾性膜変形問題

ここでは、弾性変形問題から始めましょう。以下の *add_isotropic_linearized_elasticity_brick* によって追加されている定義済みのブリックを使用します。対応する項は以下の通りです。

$$\int_\Omega (\lambda^* \mathrm{div}(u) I + 2\mu\bar{\varepsilon}(u)) : \bar{\varepsilon}(\delta_u) dx,$$

この追加を接線線形システムに対して行います。このモデルブリックを使用するために、 Lamé 係数に対応するデータは、最初にモデルに追加する必要があります。ここでは、 Lamé 係数は、領域に対して一定です。ただし、一部の非定数データを定義することもできます。また、この定義済みのブリックを使用する代わりに、弱形式言語項 *add_linear_term(md mim, "lambda*(Div_u*Div_Test_u) + mu*((Grad_u + Grad_u'):Grad_Test_u)"* を使用することもできます。

連成項

$$\int_\Omega (\beta\theta I) : \bar{\varepsilon}(\delta_u) dx,$$

は定義済みのブリックはなく、弱形式言語項 *add_linear_term(md mim, "beta*theta*Div_Test_u)"* を直接使用します。弱形式言語の詳細については、 *ud-gasm-high* を参照してください。基本的に、アセンブリ文字列は、各 Gauss 点で実行される最適化されたアセンブリ命令のリストでコンパイルされます。

以下のプログラムは、全体の弾性変形方程式を考慮に入れることができます。左側の境界に Dirichlet 条件を規定するために、既定のブリックを使用します。Dirichlet 条件を定義するいくつかのオプションがあります (*ud-model-Dirichlet* を参照)。

C++

```
md.add_initialized_scalar_data("cmu", cmu);
md.add_initialized_scalar_data("clambdastar", clambdastar);
md.add_initialized_scalar_data("T0", T0);
getfem::add_isotropic_linearized_elasticity_brick
  (md, mim, "u", "clambdastar", "cmu");
getfem::add_Dirichlet_condition_with_multipliers
  (md, mim, "u", bgeot::dim_type(elements_degree-1), LEFT_BOUND);
md.add_initialized_fixed_size_data("Fdata",
 ↪  base_small_vector(F*epsilon,0.));
getfem::add_source_term_brick(md, mim, "u", "Fdata", RIGHT_BOUND);

md.add_initialized_scalar_data("beta", alpha_th*E/(1-2*nu));
getfem::add_linear_term(md, mim, "beta*(T0-theta)*Div_Test_u");
```

Python

```
md.add_initialized_data('cmu', [cmu])
md.add_initialized_data('clambdastar', [clambdastar])
md.add_initialized_data('T0', [T0])
md.add_isotropic_linearized_elasticity_brick(mim, 'u', 'clambdastar',
 ↪  'cmu')

md.add_Dirichlet_condition_with_multipliers(mim, 'u',
 ↪  elements_degree-1, LEFT_BOUND)
md.add_initialized_data('Fdata', [F*epsilon, 0])
md.add_source_term_brick(mim, 'u', 'Fdata', RIGHT_BOUND)

md.add_initialized_data('beta', [alpha_th*E/(1-2*nu)])
md.add_linear_term(mim, 'beta*(T0-theta)*Div_Test_u')
```

Scilab

```
gf_model_set(md, 'add initialized data', 'cmu', [cmu]);
gf_model_set(md, 'add initialized data', 'clambdastar',
 ↪  [clambdastar]);
gf_model_set(md, 'add initialized data', 'T0', [T0]);
gf_model_set(md, 'add isotropic linearized elasticity brick', mim,
 ↪  'u', 'clambdastar', 'cmu');

gf_model_set(md, 'add Dirichlet condition with multipliers', mim, 'u',
 ↪  elements_degree-1, LEFT_BOUND);
gf_model_set(md, 'add initialized data', 'Fdata', [F*epsilon, 0]);
gf_model_set(md, 'add source term brick', mim, 'u', 'Fdata',
 ↪  RIGHT_BOUND);

gf_model_set(md, 'add initialized data', 'beta',
 ↪  [alpha_th*E/(1-2*nu)]);
gf_model_set(md, 'add linear term', mim,
 ↪  'beta*(T0-theta)*Div_Test_u');
```

Matlab

```
gf_model_set(md, 'add initialized data', 'cmu', [cmu]);
gf_model_set(md, 'add initialized data', 'clambdastar',
 ↪  [clambdastar]);
gf_model_set(md, 'add initialized data', 'T0', [T0]);
gf_model_set(md, 'add isotropic linearized elasticity brick', mim,
 ↪  'u', 'clambdastar', 'cmu');

gf_model_set(md, 'add Dirichlet condition with multipliers', mim, 'u',
 ↪  elements_degree-1, LEFT_BOUND);
gf_model_set(md, 'add initialized data', 'Fdata', [F*epsilon, 0]);
gf_model_set(md, 'add source term brick', mim, 'u', 'Fdata',
 ↪  RIGHT_BOUND);

gf_model_set(md, 'add initialized data', 'beta',
 ↪  [alpha_th*E/(1-2*nu)]);
gf_model_set(md, 'add linear term', mim,
 ↪  'beta*(T0-theta)*Div_Test_u');
```

4.3.9 電位問題

同様に、以下のプログラムは、電位方程式を記述しています。電気伝導率 σ と弱形式言語の項の定義方法に注意してください。

C++

```
std::string sigmaeps = "(eps/(rho_0*(1+alpha*(theta-T0))))";
md.add_initialized_scalar_data("eps", epsilon);
md.add_initialized_scalar_data("rho_0", rho_0);
md.add_initialized_scalar_data("alpha", alpha);
getfem::add_nonlinear_term
  (md, mim, sigmaeps+"*(Grad_V.Grad_Test_V)");
getfem::add_Dirichlet_condition_with_multipliers
  (md, mim, "V", bgeot::dim_type(elements_degree-1), RIGHT_BOUND);
md.add_initialized_scalar_data("DdataV", 0.1);
getfem::add_Dirichlet_condition_with_multipliers
  (md, mim, "V", bgeot::dim_type(elements_degree-1), LEFT_BOUND,
  ↪  "DdataV");
```

Python

```
sigmaeps = '(eps/(rho_0*(1+alpha*(theta-T0))))'
md.add_initialized_data('eps', [epsilon])
md.add_initialized_data('rho_0', [rho_0])
md.add_initialized_data('alpha', [alpha])
md.add_nonlinear_term(mim, sigmaeps+'*(Grad_V.Grad_Test_V)')
md.add_Dirichlet_condition_with_multipliers(mim, 'V',
 ↪  elements_degree-1, RIGHT_BOUND)
md.add_initialized_data('DdataV', [0.1])
md.add_Dirichlet_condition_with_multipliers(mim, 'V',
 ↪  elements_degree-1, LEFT_BOUND, 'DdataV')
```

Scilab

```
sigmaps = '(eps/(rho_0*(1+alpha*(theta-T0))))';
gf_model_set(md, 'add initialized data', 'eps', [epsilon]);
gf_model_set(md, 'add initialized data', 'rho_0', [rho_0]);
gf_model_set(md, 'add initialized data', 'alpha', [alpha]);
gf_model_set(md, 'add nonlinear term', mim,
 ↪  sigmaeps+'*(Grad_V.Grad_Test_V)');
gf_model_set(md, 'add Dirichlet condition with multipliers', mim, 'V',
 ↪  elements_degree-1, RIGHT_BOUND);
gf_model_set(md, 'add initialized data', 'DdataV', [0.1]);
gf_model_set(md, 'add Dirichlet condition with multipliers', mim, 'V',
 ↪  elements_degree-1, LEFT_BOUND, 'DdataV');
```

Matlab

```
sigmaps = '(eps/(rho_0*(1+alpha*(theta-T0))))';
gf_model_set(md, 'add initialized data', 'eps', [epsilon]);
gf_model_set(md, 'add initialized data', 'rho_0', [rho_0]);
gf_model_set(md, 'add initialized data', 'alpha', [alpha]);
gf_model_set(md, 'add nonlinear term', mim, [sigmaeps
 ↪  '*(Grad_V.Grad_Test_V)']);
gf_model_set(md, 'add Dirichlet condition with multipliers', mim, 'V',
 ↪  elements_degree-1, RIGHT_BOUND);
gf_model_set(md, 'add initialized data', 'DdataV', [0.1]);
gf_model_set(md, 'add Dirichlet condition with multipliers', mim, 'V',
 ↪  elements_degree-1, LEFT_BOUND, 'DdataV');
```

4.3.10 熱問題

ここで、熱問題を記述するプログラムは次の通りです。

C++

```
md.add_initialized_scalar_data("kaeps", kappa*epsilon);
getfem::add_generic_elliptic_brick(md, mim, "theta", "kaeps");
md.add_initialized_scalar_data("D2", D*2);
md.add_initialized_scalar_data("D2airt", air_temp*D*2);
getfem::add_mass_brick(md, mim, "theta", "D2");
getfem::add_source_term_brick(md, mim, "theta", "D2airt");

getfem::add_nonlinear_term
  (md, mim, "-"+sigmaeps+"*Norm_sqr(Grad_V)*Test_theta");
```

Python

```
md.add_initialized_data('kaeps', [kappa*epsilon])
md.add_generic_elliptic_brick(mim, 'theta', 'kaeps')
md.add_initialized_data('D2', [D*2])
md.add_initialized_data('D2airt', [air_temp*D*2])
md.add_mass_brick(mim, 'theta', 'D2')
md.add_source_term_brick(mim, 'theta', 'D2airt')

md.add_nonlinear_term(mim,
 ↪  '-'+sigmaeps+'*Norm_sqr(Grad_V)*Test_theta')
```

Scilab

```
gf_model_set(md, 'add initialized data', 'kaeps', [kappa*epsilon]);
gf_model_set(md, 'add generic elliptic brick', mim, 'theta', 'kaeps');
gf_model_set(md, 'add initialized data', 'D2', [D*2]);
gf_model_set(md, 'add initialized data', 'D2airt', [air_temp*D*2]);
gf_model_set(md, 'add mass brick', mim, 'theta', 'D2');
gf_model_set(md, 'add source term brick', mim, 'theta', 'D2airt');

gf_model_set(md, 'add nonlinear term', mim,
 ↪  '-'+sigmaeps+'*Norm_sqr(Grad_V)*Test_theta');
```

Matlab

```
gf_model_set(md, 'add initialized data', 'kaeps', [kappa*epsilon]);
gf_model_set(md, 'add generic elliptic brick', mim, 'theta', 'kaeps');
gf_model_set(md, 'add initialized data', 'D2', [D*2]);
gf_model_set(md, 'add initialized data', 'D2airt', [air_temp*D*2]);
gf_model_set(md, 'add mass brick', mim, 'theta', 'D2');
gf_model_set(md, 'add source term brick', mim, 'theta', 'D2airt');

gf_model_set(md, 'add nonlinear term', mim, ['-' sigmaeps
 ↪  '*Norm_sqr(Grad_V)*Test_theta']);
```

4.3.11 モデルの求解

モデルを正しく定義したら、次のようにして簡単に解くことができます。

C++
```
gmm::iteration iter(1E-9, 1, 100);
getfem::standard_solve(md, iter);
```

Python
```
md.solve('max_res', 1E-9, 'max_iter', 100, 'noisy')
```

Scilab
```
gf_model_get(md, 'solve', 'max_res', 1E-9, 'max_iter', 100, 'noisy');
```

Matlab
```
gf_model_get(md, 'solve', 'max_res', 1E-9, 'max_iter', 100, 'noisy');
```

問題は大域的に非線形であるため、Newton 法を用いて問題を反復的に解きます。数回のイテレーションが必要です (この場合は約 4 です)。

4.3.12 2 ステップでのモデルの求解

別の解き方として、最初に熱と電位の問題を解くこともできます。今回のモデルでは、熱および電位は変形に依存しません。熱と電位の問題を解いてから変形の問題を解くには、次のように実行します。

C++
```
gmm::iteration iter(1E-9, 1, 100);
md.disable_variable("u");
getfem::standard_solve(md, iter);
md.enable_variable("u");
md.disable_variable("theta");
md.disable_variable("V");
iter.init();
getfem::standard_solve(md, iter);
```

Python
```
md.disable_variable('u')
md.solve('max_res', 1E-9, 'max_iter', 100, 'noisy')
md.enable_variable('u')
md.disable_variable('theta')
md.disable_variable('V')
md.solve('max_res', 1E-9, 'max_iter', 100, 'noisy')
```

Scilab
```
gf_model_set(md, 'disable variable', 'u');
gf_model_get(md, 'solve', 'max_res', 1E-9, 'max_iter', 100, 'noisy');
gf_model_set(md, 'enable variable', 'u');
gf_model_set(md, 'disable variable', 'theta');
gf_model_set(md, 'disable variable', 'V');
gf_model_get(md, 'solve', 'max_res', 1E-9, 'max_iter', 100, 'noisy');
```

Matlab
```
gf_model_set(md, 'disable variable', 'u');
gf_model_get(md, 'solve', 'max_res', 1E-9, 'max_iter', 100, 'noisy');
gf_model_set(md, 'enable variable', 'u');
gf_model_set(md, 'disable variable', 'theta');
gf_model_set(md, 'disable variable', 'V');
gf_model_get(md, 'solve', 'max_res', 1E-9, 'max_iter', 100, 'noisy');
```

4.3.13 解のエクスポート/可視化

以上で有限要素問題が解けました。図のように解をプロットすることができます。C++ および Python プログラムでは、外部のグラフィカルポストプロセッサを使用する必要があることに注意してください。また、汎用補間を使用して任意の数量を後処理できることにも注意してください (後述の *ga_interpolation_Lagrange_fem* を参照)。また、複雑なエクスポートやスライスの作成も可能です (*ud-export* をご覧ください)。

C++

```
plain_vector U(mfu.nb_dof()); gmm::copy(md.real_variable("u"), U);
plain_vector V(mft.nb_dof()); gmm::copy(md.real_variable("V"), V);
plain_vector THETA(mft.nb_dof());
 ↪  gmm::copy(md.real_variable("theta"),THETA);
plain_vector VM(mfvm.nb_dof());
getfem::compute_isotropic_linearized_Von_Mises_or_Tresca
  (md, "u", "clambdastar", "cmu", mfvm, VM, false);
plain_vector CO(mfvm.nb_dof() * 2);
getfem::ga_interpolation_Lagrange_fem(md, "-"+sigmaeps+"*Grad_V",
 ↪  mfvm, CO);

getfem::vtk_export exp("displacement_with_von_mises.vtk", false);
exp.exporting(mfu);
exp.write_point_data(mfu, U, "elastostatic displacement");
exp.write_point_data(mfvm, VM, "Von Mises stress");
cout << "\nYou can view solutions with for instance:\n\nmayavi2 "
  "-d displacement_with_von_mises.vtk -f WarpVector -m Surface\n" <<
   ↪  endl;

getfem::vtk_export exp2("temperature.vtk", false);
exp2.exporting(mft);
exp2.write_point_data(mft, THETA, "Temperature");
cout << "mayavi2 -d temperature.vtk -f WarpScalar -m Surface\n" <<
 ↪  endl;

getfem::vtk_export exp3("electric_potential.vtk", false);
exp3.exporting(mft);
exp3.write_point_data(mft, V, "Electric potential");
cout << "mayavi2 -d electric_potential.vtk -f WarpScalar -m Surface\n"
     << endl;
}
```

Python

```
U = md.variable('u')
V = md.variable('V')
THETA = md.variable('theta')
VM = md.compute_isotropic_linearized_Von_Mises_or_Tresca('u',
 ↪  'clambdastar', 'cmu', mfvm)
CO = np.reshape(md.interpolation('-'+sigmaeps+'*Grad_V', mfvm), (2,
 ↪  mfvm.nbdof()), 'F')

mfvm.export_to_vtk('displacement_with_von_mises.vtk', mfvm,
                   VM, 'Von Mises Stresses', mfu, U, 'Displacements')
print ('You can view solutions with for instance:')
print ('mayavi2 -d displacement_with_von_mises.vtk -f WarpVector -m
 ↪  Surface')
mft.export_to_vtk('temperature.vtk', mft, THETA, 'Temperature')
print ('mayavi2 -d temperature.vtk -f WarpScalar -m Surface')
mft.export_to_vtk('electric_potential.vtk', mft, V, 'Electric
 ↪  potential')
print ('mayavi2 -d electric_potential.vtk -f WarpScalar -m Surface')
```

Scilab

```
U = gf_model_get(md, 'variable', 'u');
V = gf_model_get(md, 'variable', 'V');
THETA = gf_model_get(md, 'variable', 'theta');
VM = gf_model_get(md,
 ↪  'compute_isotropic_linearized_Von_Mises_or_Tresca', 'u',
 ↪  'clambdastar', 'cmu', mfvm);
CO = matrix(gf_model_get(md, 'interpolation', '-'+sigmaeps+'*Grad_V',
 ↪  mfvm), [2 gf_mesh_fem_get(mfvm, 'nbdof')]);

hh = scf(2);
hh.color_map = jetcolormap(255);
subplot(3,1,1);
gf_plot(mfvm, VM, 'mesh', 'off', 'deformed_mesh','off', 'deformation',
 ↪  U, 'deformation_mf', mfu, 'deformation_scale', 100, 'refine', 8);
colorbar(min(VM),max(VM));
title('Von Mises stress in N/cm^2 (on the deformed configuration,
 ↪  scale factor x100)');
subplot(3,1,2);
drawlater;
gf_plot(mft, V, 'mesh', 'off', 'deformed_mesh','off', 'deformation',
 ↪  U, 'deformation_mf', mfu, 'deformation_scale', 100, 'refine', 8);
colorbar(min(V),max(V));
gf_plot(mfvm, CO, 'quiver', 'on', 'quiver_density', 0.1, 'mesh',
 ↪  'off', 'deformed_mesh','off', 'deformation_mf', mfu, ...
        'deformation', U, 'deformation_scale', 100, 'refine', 8);
title('Electric potential in Volt (on the deformed configuration,
 ↪  scale factor x100)');
drawnow;
subplot(3,1,3);
gf_plot(mft, THETA, 'mesh', 'off', 'deformed_mesh','off',
 ↪  'deformation', U, 'deformation_mf', mfu, 'deformation_scale', 100,
 ↪  'refine', 8);
colorbar(min(THETA),max(THETA));
title('Temperature in °C (on the deformed configuration, scale factor
 ↪  x100)');
```

Matlab

```
U = gf_model_get(md, 'variable', 'u');
V = gf_model_get(md, 'variable', 'V');
THETA = gf_model_get(md, 'variable', 'theta');
VM = gf_model_get(md,
 ↪  'compute_isotropic_linearized_Von_Mises_or_Tresca', 'u',
 ↪  'clambdastar', 'cmu', mfvm);
CO = reshape(gf_model_get(md, 'interpolation', ['-' sigmaeps
 ↪  '*Grad_V'], mfvm), [2 gf_mesh_fem_get(mfvm, 'nbdof')]);

figure(2);
subplot(3,1,1);
gf_plot(mfvm, VM, 'mesh', 'off', 'deformed_mesh','off', 'deformation',
 ↪  U, 'deformation_mf', mfu, 'deformation_scale', 100, 'refine', 8);
colorbar;
title('Von Mises stress in N/cm^2 (on the deformed configuration,
 ↪  scale factor x100)');
subplot(3,1,2);
hold on;
gf_plot(mft, V, 'mesh', 'off', 'deformed_mesh','off', 'deformation',
 ↪  U, 'deformation_mf', mfu, 'deformation_scale', 100, 'refine', 8);
colorbar;
gf_plot(mfvm, CO, 'quiver', 'on', 'quiver_density', 0.1, 'mesh',
 ↪  'off', 'deformed_mesh','off', 'deformation', U, 'deformation_mf',
 ↪  ...
        mfu, 'deformation_scale', 100, 'refine', 8);
colorbar;
title('Electric potential in Volt (on the deformed configuration,
 ↪  scale factor x100)');
hold off;
subplot(3,1,3);
gf_plot(mft, THETA, 'mesh', 'off', 'deformed_mesh','off',
 ↪  'deformation', U, 'deformation_mf', mfu, 'deformation_scale', 100,
 ↪  'refine', 8);
colorbar;
title('Temperature in ?C (on the deformed configuration, scale factor
 ↪  x100)');
```

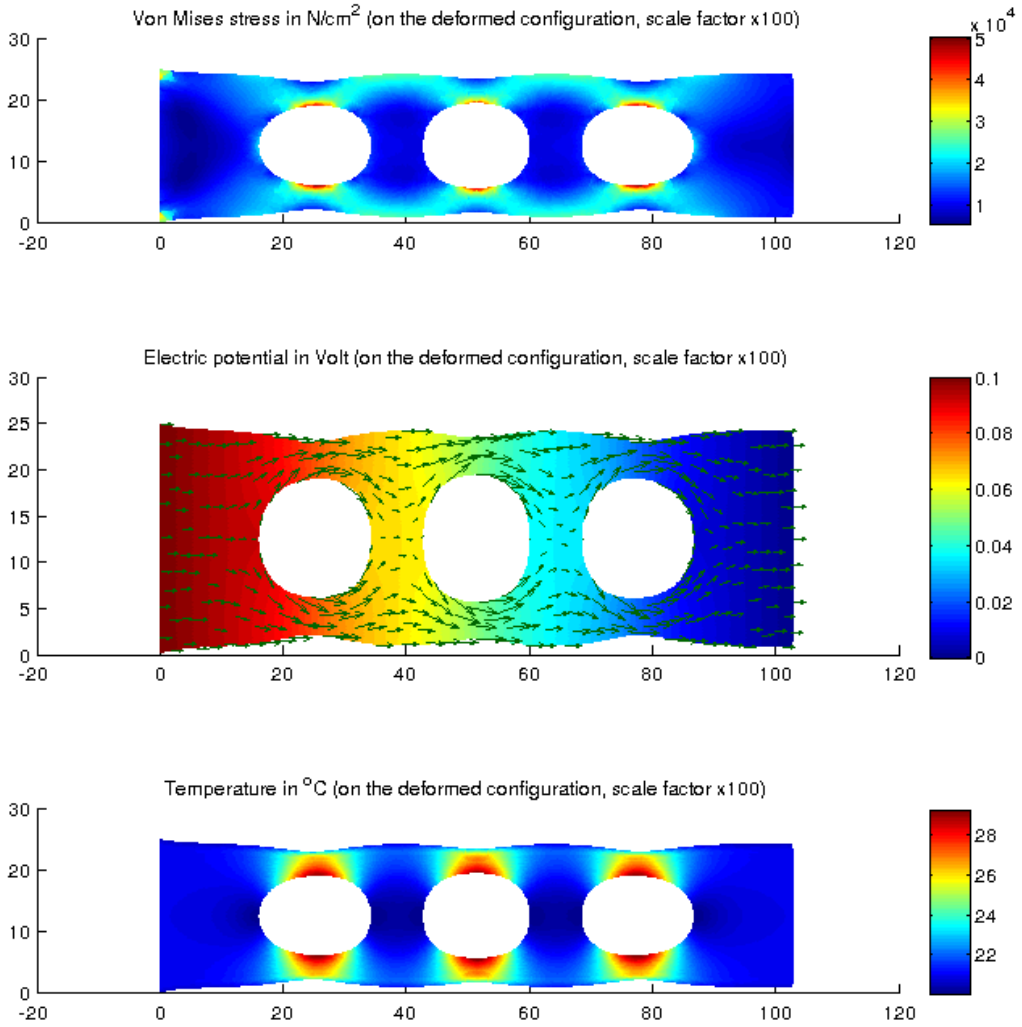


Figure 4.2: 解のプロット。

Chapter 5

接触する車輪の例 (2つのメッシュ間の組み立て、変換、固定サイズ変数の使用)

この例では、変形可能な「車輪」が変形可能な基礎と接触します。ここでは python インターフェイスを使用しますが、別のインターフェイスまたは C++ にこのプログラムを訳すのは簡単です (前の例を参照してください)。完全なプログラム *demo_wheel_contact.py* がディレクトリ *interface/tests/python* にあります。

5.1 問題の設定

$\Omega^1 \subset \mathbb{R}^2$ は 2 次元の車輪の基準配置であり、 $\Omega^2 \subset \mathbb{R}^2$ は変形可能な基礎の基準配置です。これら 2 つの (線形弾性) 体の微小変形とそれらの間の接触を考慮します。また、車輪の周縁は剛であり、車輪に垂直方向の力がはたらくとします。

5.2 プログラムの作成

Getfem をロードし、問題のパラメータを設定することから始めましょう。

```
import getfem as gf
import numpy as np

E = 21E6                          # Young Modulus (N/cm^2)
nu = 0.3                          # Poisson ratio
clambda = E*nu/((1+nu)*(1-2*nu)) # First Lame coefficient (N/cm^2)
cmu = E/(2*(1+nu))                # Second Lame coefficient (N/cm^2)
clambdastar = 2*clambda*cmu/(clambda+2*cmu) # Lame coefficient for Plane stress
↪  (N/cm^2)
applied_force = 1E7               # Force at the hole boundary (N)

h = 1                       # Approximate mesh size
elements_degree = 2         # Degree of the finite element methods
gamma0 = 1./E;              # Augmentation parameter for the augmented Lagrangian
```

5.2.1 メッシュ生成

車輪の半径は 15cm、周縁 8cm のもので、車輪は厚さ 10cm の変形可能な基礎の上にあると考えます。*GetFEM++* の実験的なメッシャーを使用して、車輪のメッシュを生成します。基礎のメッシュに関しては、構造化されたメッシュを構築します (python インタフェースのメッシュオブジェクトのドキュメントを参照してください)。

```
mo1 = gf.MesherObject('ball', [0., 15.], 15.)
mo2 = gf.MesherObject('ball', [0., 15.], 8.)
mo3 = gf.MesherObject('set minus', mo1, mo2)
gf.util('trace level', 2)   # No trace for mesh generation
mesh1 = gf.Mesh('generate', mo3, h, 2)

mesh2 =
↪  gf.Mesh('import','structured','GT="GT_PK(2,1)";SIZES=[30,10];NOISED=0;NSUBDIV=[%d,%d];'
↪  % (int(30/h)+1, int(10/h)+1));
mesh2.translate([-15.,-10.])
```

結果は図のとおりです。

5.2.2 境界の選択

(荷重や剛体であることを仮定する) 周縁の境界 、車輪の接触境界、およびクランプしたと仮定する基礎の底部境界などの境界条件を設定するために、さまざまな境界を選択する必要があります。

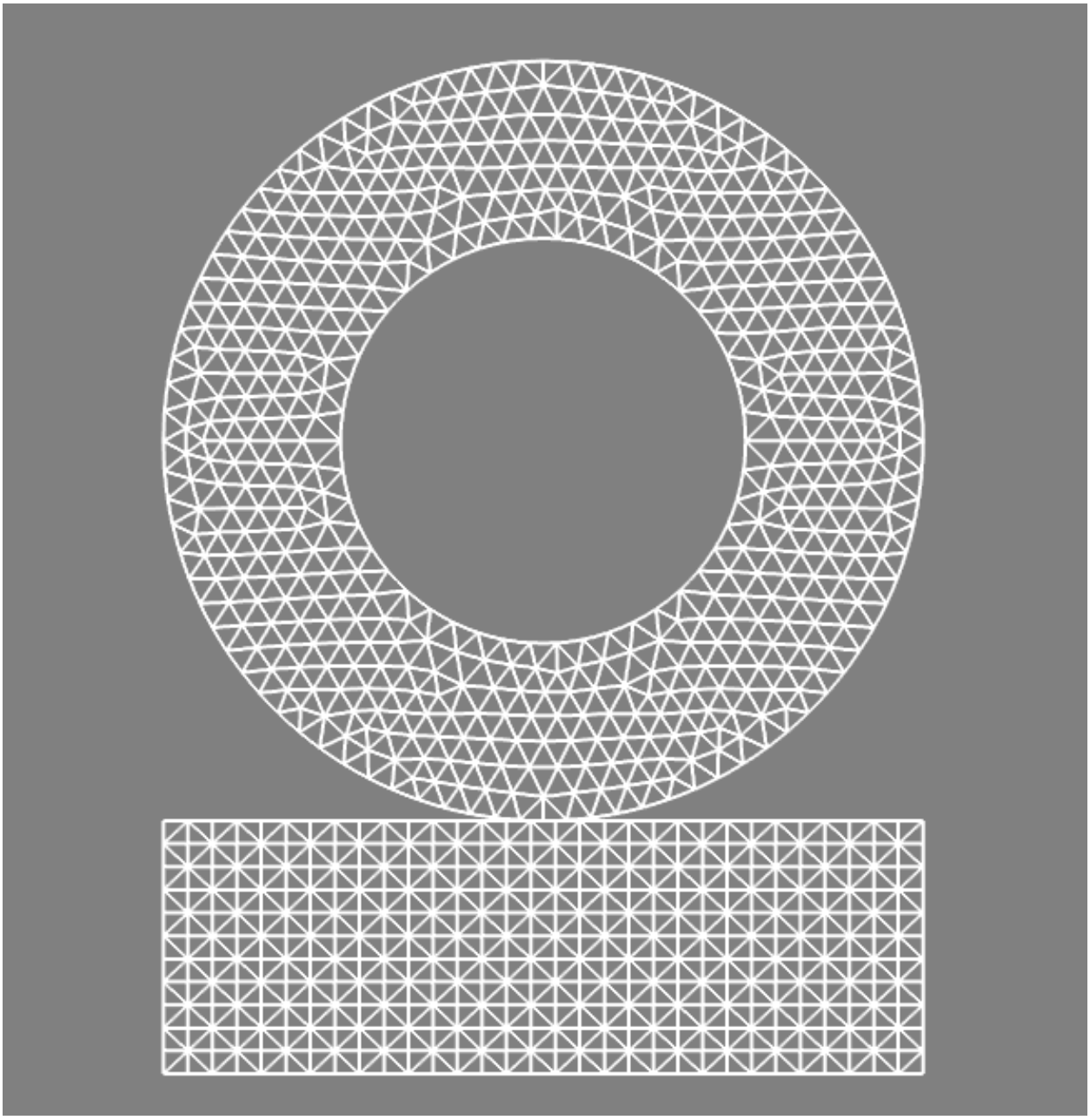
```
fb1 = mesh1.outer_faces_in_box([-8.1, 6.9], [8.1, 23.1])     # Boundary of the hole
fb2 = mesh1.outer_faces_with_direction([0., -1.], np.pi/4.5) # Contact boundary of the
↪  wheel
fb3 = mesh2.outer_faces_with_direction([0., -1.], 0.01)      # Bottom boundary of the
↪  foundation

HOLE_BOUND=1; CONTACT_BOUND=2; BOTTOM_BOUND=3;

mesh1.set_region(HOLE_BOUND, fb1)
mesh1.set_region(CONTACT_BOUND, fb2)
mesh1.region_subtract(CONTACT_BOUND, HOLE_BOUND)
mesh2.set_region(BOTTOM_BOUND, fb3)
```

コマンド *mesh1.outer_faces_with_direction([0., -1.], np.pi/4)* は、ベクトル *[0., -1.]* に対して角度 *np.pi/4* 以下の外向きの単位法線を持つすべての面を選択します。コマンド *mesh1.region_subtract(CONTACT_BOUND, HOLE_BOUND)* は接触境界のリムの面を削除します。

5.2.3 有限要素法と積分法の定義

車輪と基礎の変位をそれぞれ近似する有限要素法 *mfu1*, *mfu2* を定義します。 *mflambda* はリムの剛性を考慮した乗数を近似する有限要素法で、 *mflambda_C* は接触乗数 (接触圧力) を近似するための有限要素法です、 *mfvm1*, *mfvm2* は車輪と基礎の Von Mises 応力を補間し後処理に使用されます。 *mim1*, *mim2* は車輪と基礎のそれぞれの積分法です。

```
mfu1 = gf.MeshFem(mesh1, 2)
mfu1.set_classical_fem(elements_degree)
mflambda = gf.MeshFem(mesh1, 2)
mflambda.set_classical_fem(elements_degree-1)
mflambda_C = gf.MeshFem(mesh1, 1)
mflambda_C.set_classical_fem(elements_degree-1)
mfu2 = gf.MeshFem(mesh2, 2)
```



```
mfu2.set_classical_fem(elements_degree)
mfvm1 = gf.MeshFem(mesh1, 1)
mfvm1.set_classical_discontinuous_fem(elements_degree)
mfvm2 = gf.MeshFem(mesh2, 1)
mfvm2.set_classical_discontinuous_fem(elements_degree)
mim1 = gf.MeshIm(mesh1, pow(elements_degree,2))
mim2 = gf.MeshIm(mesh2, pow(elements_degree,2))
```

5.2.4 モデルの定義

実数モデルを使用して、変位を表す 2 つの変数を宣言します。

```
md=gf.Model('real');
md.add_fem_variable('u1', mfu1)       # Displacement of the structure 1
md.add_fem_variable('u2', mfu2)       # Displacement of the structure 2
```

5.2.5 線形弾性ブリック

Lamé 係数をモデルのデータとして追加し、車輪と基礎に線形弾性ブリックを追加します。

```
md.add_initialized_data('cmu', [cmu])
md.add_initialized_data('clambdastar', [clambdastar])
md.add_isotropic_linearized_elasticity_brick(mim1, 'u1', 'clambdastar', 'cmu')
md.add_isotropic_linearized_elasticity_brick(mim2, 'u2', 'clambdastar', 'cmu')
```

5.2.6 基礎の下部の固定条件

例えば、次のブリックを追加し乗算することで、基礎の底面の変位をなくすように設定できます。

```
md.add_Dirichlet_condition_with_multipliers(mim2, 'u2', elements_degree-1,
↪  BOTTOM_BOUND)
```

5.2.7 接触条件 (補間変換の使用)

ここで、2 つの構造の間に接触条件を設定する方法を見てみましょう。定義済みのブリックを使用することもできますが (微小変形/微小滑りは *ud-model-contact-friction* そして大変形/大滑りは *ud-model-contact-friction-large* を参照してください)、ここでは拡張 Lagrangian 式と変換の補間を使用して接触条件を直接規定する方法について解説します。

微小変形接触では、接触面の点の対応を基準配置に記述する必要があります。この点については開発が進んでおらず、単純な近似値を使用します。

車輪の接触境界がスレーブ 1 であることを考慮して、車輪の接触境界から基礎の接触境界へのの変換を記述しなければなりません。これは、基礎の接触境界が垂直座標の消去に対応するため非常に簡単です。そこで、変換を定義します。

$$X \longmapsto (X(1), 0)$$

ここで X は点の座標ベクトルです。この変換をコマンドを使用してモデルに追加します。

```
md.add_interpolate_transformation_from_expression('Proj1', mesh1, mesh2, '[X(1);0]')
```

これにより、車輪のメッシュから基礎のメッシュへの変換を弱形式言語で表現することができます。これは非常に単純な定数式ですが、データまたはモデルの変数に応じて、より複雑な式を使用できます。変換の式がモデルの変数に依存する場合、接線線形システムでは、この依存関係が自動的に考慮されます (詳細については *ud-gasm-high-transf* を参照してください。また、有限すべり接触に対応し接触境界間の対応を自動的に検索する変換も *GetFEM++* にあります。 (*ud-model-contact-friction-large-hlgav* を参照)。

定義された変換を使用し、拡張 Lagrangian 式を使用した積分接触条件を記述することができます (詳細については *ud-model-contact-friction* を参照)。対応する項 (弱定式化の残りの部分に追加されます) は以下になります。

$$\cdots + \int_{\Gamma_c} \lambda_N(X)(\delta_{u^1}(X) - \delta_{u^2}(\Pi(X))) \cdot n d\Gamma$$
$$- \int_{\Gamma_c} \left(\lambda_N(X) + \left(\lambda_N(X) + \frac{1}{h_T \gamma_0}((X + u^1(X)) \cdot n - (\Pi(X) - u^2(\Pi(X))) \cdot n \right)_- \right) \delta_{\lambda_N}(X) d\Gamma = 0 \quad \forall \delta_{\lambda_N}, \forall \delta_{u^1}, \forall \delta_{u^2},$$

ここで Γ_c はスレーブ接触境界です。 λ_N は接触乗数 (接触圧) です。 h_T は要素の半径です。 Π は変換です。 n ($n = (0, 1)$) はマスター接触境界への外向き法線ベクトルです。 γ_0 は拡張パラメータです。 $(\cdot)_- : I\!R \to I\!R_+$ は負の部分です。 $\delta_{\lambda_N}, \delta_{u^1}, \delta_{u^2}$ はそれぞれ λ_N, u^1, u^2 に対応する試行関数です。

弱形式言語を使用すると、次の方法で接触条件を追加できます。

```
md.add_initialized_data('gamma0', [gamma0])
md.add_filtered_fem_variable('lambda1', mflambda_C, CONTACT_BOUND)
md.add_nonlinear_term(mim1, 'lambda1*(Test_u1.[0;1])'
                      '-lambda1*(Interpolate(Test_u2,Proj1).[0;1])', CONTACT_BOUND)
md.add_nonlinear_term(mim1, '-(gamma0*element_size)'
          '*(lambda1 + neg_part(lambda1+(1/(gamma0*element_size))'
          '*((u1-Interpolate(u2,Proj1)+X-Interpolate(X,Proj1)).[0;1])))*Test_lambda1',
          ↪  CONTACT_BOUND);
```

5.2.8 リムの剛性と垂直力の処理

次にリムの剛性を設定する必要があります。リムの垂直方向の変位が何になるかは未知のため、剛性は非標準の状態です。そこで、垂直変位のための未知変数を追加します。サイズ 1 の (つまり有限要素フィールドではない) 固定サイズの変数 *alpha_D* を追加します。

```
md.add_variable('alpha_D', 1)
```

リム境界上の変位を規定する乗数も必要です。

```
md.add_filtered_fem_variable('lambda_D', mflambda, HOLE_BOUND)
```

この乗数は垂直方向の変位を $(0, -\alpha_D)$ とするために必要な境界応力を表します。乗数をリムの境界上で積分すると適用したい垂直方向の力になるように設定をします。追加される弱定式項の残りの対応部分の項についても見てみましょう。

$$\cdots + \int_{\Gamma_D} -\lambda_D.\delta_{u^1} + ((0, \alpha_D) - u^1).\delta_{\lambda_D} + (\lambda_D.(0, 1) + F)\delta_{\alpha_D} d\Gamma = 0,$$

ここで Γ_D はリムの境界であり、 F は力の密度です。

これを弱形式言語でモデルに追加します。

```
md.add_filtered_fem_variable('lambda_D', mflambda, HOLE_BOUND)
md.add_initialized_data('F', [applied_force/(8*2*np.pi)])
md.add_linear_term(mim1, '-lambda_D.Test_u1 + (alpha_D*[0;1]-u1).Test_lambda_D'
      ' + (lambda_D.[0;1]+F)*Test_alpha_D', HOLE_BOUND)
```

よりロバスト性を向上させるために $alpha_D$ 上に微小のペナルティを追加することもできます。

```
md.add_linear_term(mim1, '1E-6*alpha_D*Test_alpha_D');
```

固定サイズ変数 *alpha_D* は、リム境界の各点にリンクされていることに注意してください。そのため *alpha_D* に対応する接線行列の行は、ゼロ以外の成分を多く含む可能性があります。ゆえに、固定サイズ変数の使用は慎重に行う必要があります。

5.2.9 モデルを解く

次のコードで問題を解くことができます。

```
md.solve('max_res', 1E-9, 'max_iter', 100, 'noisy')
```

設定によっては、デフォルトのものよりも基本的な線形探索を使用する方が望ましい場合もあります。

```
md.solve('max_res', 1E-9, 'max_iter', 100, 'noisy', 'lsearch', 'simplest',  'alpha
↪  min', 0.8)
```

5.2.10 解の出力

最後に、VonMises 応力の解を出力するコードを示します。

```
U1 = md.variable('u1')
U2 = md.variable('u2')
VM1 = md.compute_isotropic_linearized_Von_Mises_or_Tresca('u1', 'clambdastar', 'cmu',
↪  mfvm1)
VM2 = md.compute_isotropic_linearized_Von_Mises_or_Tresca('u2', 'clambdastar', 'cmu',
↪  mfvm2)

mfvm1.export_to_vtk('displacement_with_von_mises1.vtk', mfvm1,  VM1, 'Von Mises
↪  Stresses',
                    mfu1, U1, 'Displacements')

mfvm2.export_to_vtk('displacement_with_von_mises2.vtk', mfvm2,  VM2, 'Von Mises
↪  Stresses',
                   mfu2, U2, 'Displacements')
# You can view solutions with for instance:
# mayavi2 -d displacement_with_von_mises1.vtk -f WarpVector -m Surface -d
↪  displacement_with_von_mises2.vtk -f WarpVector -m Surface
```

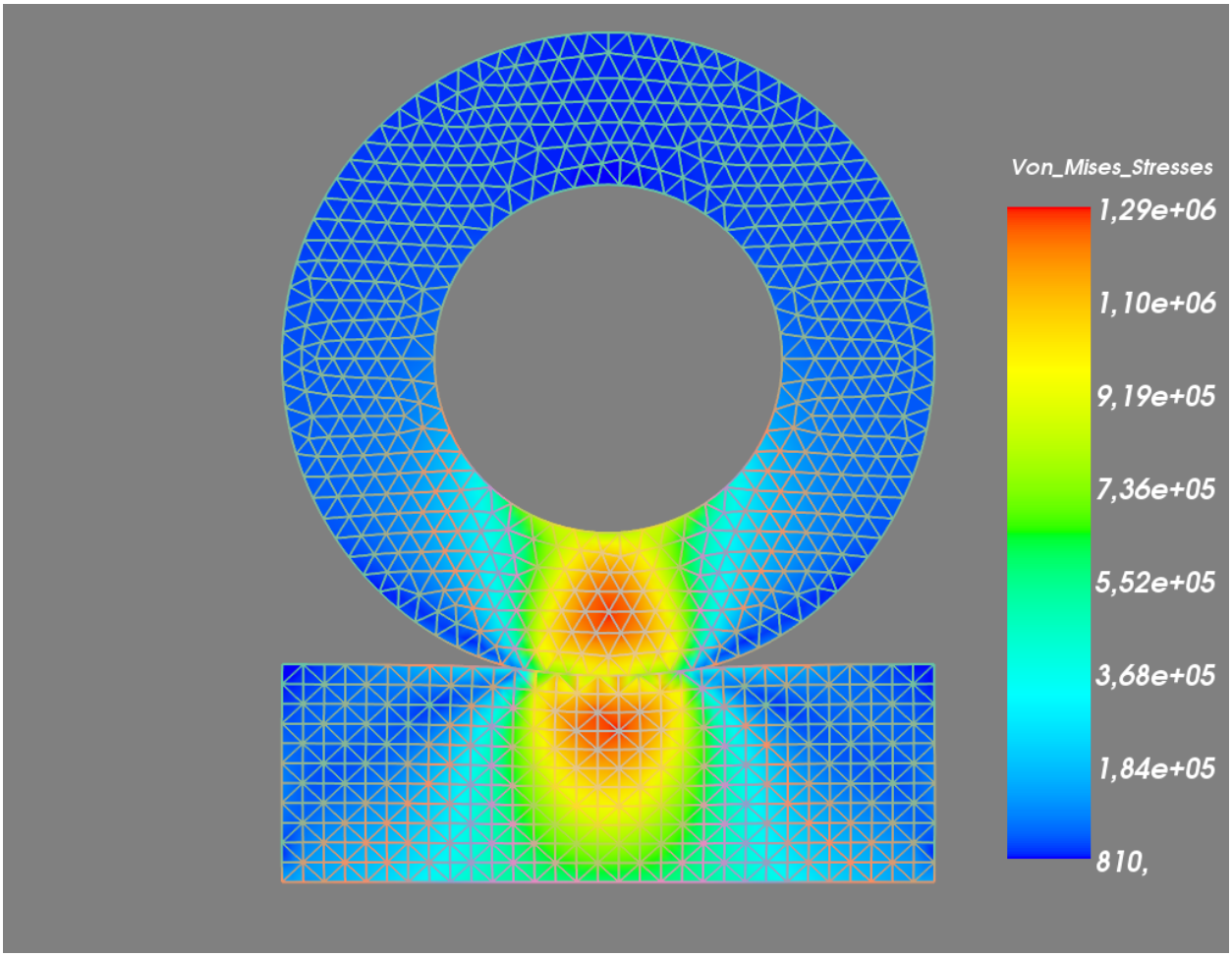


Figure 5.1: 解のプロット。

Description of the Project

リリース *5.3*

Yves Renard, Julien Pommier, Konstantinos Poulios

2019 年 03 月 26 日

Contents

Chapter 1

前書き

この文書の目的は、*GetFEM++* 内部の詳細を紹介することです。ユーザードキュメントに書いていない開発者にとって便利な情報を提供します。また、今後の *GetFEM++* 開発のための主な展望を説明することも目的としています。行わなければならない変更の一覧と主なタスクは Savannah の https://savannah.nongnu.org/task/?group=getfem にあります。

GetFEM++ プロジェクトは、オープンソースの汎用的な有限要素ライブラリの開発を中心に扱っています。偏微分方程式 (PDE) で記述されたシステムのモデル化のための数値計算コードを容易に構築できる有限要素フレームワークを提供することを目標としてます。提供されている複数の手法の切り替えが可能な限り柔軟にできるように特に注意が払われています。

従来の有限要素コードと比較して、それを可能にしている最大のポイントは、微分方程式モデルと有限要素法の記述を完全に分離したことです。また、(厳密または近似) 積分法、(線形または非線形) 幾何変換、および参照要素に記述されている任意の次数の有限要素法を分離しています。 *GetFEM++* により汎用的な有限要素コードを構築することができます。有限要素、積分法、メッシュの次元はパラメータとしてとても簡単に変更することができます。これにより、大規模な範囲での実験が可能です。配布物の `tests` ディレクトリに多数の例があります。

また、新しい有限要素法の追加を可能な限り単純にすることも目標としています。標準的なものであれば, 有限要素の形状関数と参照要素の自由度の関係を記述すれば十分です。Hermite 要素、区分多項式、非多項式、ベクトル要素、および XFem の拡張が提供されています。定義可能な手法の例を次に示します。任意の ji 次元と寸法の simplices の P_k 、 Q_k 平行六面体、 P_1 、 P_2 気泡関数、Hermite 要素、階層ベースの要素 (マルチグリッド法インスタンスのメソッド)、不連続 P_k または Q_k ‘、XFem、Argyris、HCT、Raviart-Thomas です。

このライブラリは, 以下のような有限要素を構築するための標準ツールも含んでいます。古典的な微分方程式, 補間法, ノルムの計算, メッシュ操作, 境界条件, メッシュからスライスを抽出するなどの後処理ツール...

GetFEM++ プロジェクトは、グラフィックインターフェイスを使用して構造力学計算をすることができる有限要素コードを提供することは目的にしていません。このライブラリは基本的に C++ での有限要素コードのビルドが可能です。しかし、Python、Scilab および matlab インタフェースで、問題の定義、有限要素法の選択、およびグラフィカルな後処理のアプリケーションを簡単に構築することが可能です。

Lesser General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA.

Chapter 2

貢献する方法 / SavannahのGitリポジトリ

GetFEM++ は共同開発に基づいたオープンソースの有限要素ライブラリです。貢献したい場合は、プロジェクトへのメンバー参加を依頼してください。ドキュメンテーション、バグレポート、建設的なコメント、変更の提案、バグ修正、新モデルなど、あらゆる種類の貢献が歓迎されます。

コントリビューターは、もちろんですが、変更がライブラリの正しい機能に影響を与えないよう注意し、それらの変更が下位互換性の原則に従うように注意してください。

GetFEM++ 開発についてのタスクと議論のリストは、 here を参照してください。

重要 ： コントリビューターは、自分の貢献が *GetFEM++* の LGPL ライセンスに基づいて配布されることを暗黙に承諾しているとします。

GetFEM++ のメインリポジトリはフリーソフトウェア財団の software forge である Savannah にあります（ Savannah を参照）。 Savannah のプロジェクトページは Getfem on Savannah にあります。 Getfem sources on Savannah も参照してください。

2.1 ソースを入手する方法

必要なのはソースだけで貢献しないのであれば、次のコマンドを使うだけで済みます。

```
git clone https://git.savannah.nongnu.org/git/getfem.git
```

貢献をしていただけるなら、まずはじめに *GetFEM++* プロジェクトに参加してください（Savannah のアカウントを作成する必要があります）。そして、ssh キーを登録してから（ git on Savannah を参照）、次のコマンドを使用してください。

```
git clone ssh://savannah-login@git.sv.gnu.org:/srv/git/getfem.git
```

2.2 貢献する方法

master **ブランチで直接変更することはできない** ため、ファイルを変更する前に、 **開発ブランチ** を作成してください。ブランチ名は *devel-name-subject* の形式を推奨します。ここで、 name はあなたの名前またはログイン名で、 subject は変更の題名です。たとえば、ブランチ名を *devel-me-rewrite-fem-kernel* とした場合、ブランチは次のように作成します。

```
git branch devel-me-rewrite-fem-kernel
git checkout devel-me-rewrite-fem-kernel
```

最初のコマンドでブランチを作成し、2 番目のコマンドでブランチに移動します。これにより、変更を行う準備はほぼ完了です。変更のラベルを付けるために、連絡先の名前と電子メールを以下のコマンドで指定します。

```
git config --global user.name "Your Name Comes Here"
git config --global user.email you@yourdomain.example.com
```

2.3 ドキュメントの改善と誤植修正のためのブランチ

ドキュメンテーションのみに貢献したい場合は、特定のブランチを作成する必要はありません。そのために作成された `fixmisspell` ブランチをチェックアウトしてください。

```
git checkout fixmisspell
```

2.4 変更をローカルにコミット

もし *src/toto.cc* というファイルをいくつか変更したり、新しいファイルとして追加したら、ローカルのコミットは次のコマンドで行います。

```
git add src/toto.cc
git commit -m "Your extensive commit message here"
```

この段階ではコミットはあなたのローカルリポジトリで行われていますが、Savannah リポジトリでは行われません。

2.5 Savannah リポジトリに変更をプッシュ

以下のコマンドで Savannah リポジトリに変更を移すことができます。

```
git push origin devel-me-rewrite-fem-kernel
```

当然ですが *devel-me-rewrite-fem-kernel* はあなたのブランチの名前です。この段階で、あなたの変更は Savannah リポジトリのブランチ *devel-me-rewrite-fem-kernel* に登録されます。 *GetFEM++* のマスターブランチを変更することは許可されていないので、あなたの役割はここで終わりです。

2.6 *GetFEM++* のマスターブランチに変更をマージする場合は管理者に問い合わせてください。

十分なテストで変更を検証したら、管理者に *GetFEM++* に変更をマージするように問い合わせます。その際には、管理者の 1 人に直接連絡するか、 *getfem-commits@nongnu.org* に “please merge branch devel-me-rewrite-fem-kernel” というメッセージの電子メールを送ってください。その際には変更に関する短い説明をつけてください。重要：ブランチを残す必要がある場合を除き、デフォルトではブランチはマージ後に削除されます。

2.7 他のコントリビューターが行った変更をマージします

次のコマンドで実行します。

```
git pull origin master
git merge master
```

このコマンドは検証され統合された変更をマスターブランチに統合するために使用されます。マスターブランチでの変更の統合を要求する前にこのコマンドを実行することをお勧めします。

2.8 いくつかの便利な git コマンド

```
git status  : status of your repository / branch

git log --follow "filepath"   : Show all the commits modifying the specified file (and
↪  follow the eventual change of name of the file).

gitk --follow filename : same as previous but with a graphical interface
```

2.9 ドキュメントの翻訳への貢献

新しいコントリビューターがドキュメントを翻訳する際には、 Getfem translation team on Transifex に参加することを推奨します。貢献するためには、 transifex でアカウントを作成し「言語と入力フォームを要求」をクリックしてください。翻訳後 transifex クライアントを使用して、サイトから翻訳された po ファイルをプルします。Transifex サイトで手に入る API トークンが必要です。

```
cd doc/sphinx
tx pull -l <lang>
```

ネイティブ言語のコード<lang>を設定します (Currently supported languages by Sphinx are 参照)。

> **警告:** Transifex に tx push を **しないでください**。この操作はいくつかの問題があります。チームページでファイルを 1 つずつアップロードしてください。

翻訳された po ファイルをプルした後、 *doc/sphinx/Makefile.am* の LANGUAGE に<lang>を設定します。

```
LANGUAGE      = <lang>
SPHINXOPTS    = -D language=$(LANGUAGE)
```

次に html ローカライゼーションドキュメントを以下のコマンドで作成します。

```
cd doc/sphinx
make html
```

自分の言語で pdf ファイルを作成したい場合は、次のコマンドを実行します。

```
make latex
cd build/latex
make all-pdf-<lang>
```

詳細は Sphinx Internationalization を参照してください。

Chapter 3

GetFEM++ の有限要素法の記述

このセクションの目的は *GetFEM++* での FEM の記述方法を簡単に紹介することです。文書の他の部分（要素の定義、参照要素、幾何学的変換、幾何学的変換の勾配...）で使用される表記法を固定化することが主な目的です。

3.1 凸包構造

有限要素法は、要素と呼ばれる小さな凸包領域で定義されます。有限要素法を定義できる最も単純な要素は、(1 次元のシンプレックス）セグメント、他には三角形、(2 次元と 3 次元の単体）四面体、プリズム、直方体などです。 *GetFEM++* では、要素の種類 (凸包) は `bgeot_convex_structure.h` ファイルで定義されているオブジェクト `bgeot::convex_structure` で記述されています。

ここでは頂点の座標ではなく凸包の構造だけを記述します。この種類の構造は同じタイプの凸包を記述する必要はないので、この構造は自分自身を操作することはできません。このような記述子上のポインタは `bgeot::pconvex_structure` 型で宣言します。

次の関数で通常の要素タイプの記述子に対するポインタを得ることができます。

`bgeot::`**`simplex_structure`**(dim_type *d*)
: `d` 次元のシンプレックスの記述。

`bgeot::`**`parallelepiped_structure`**(dim_type *d*)
: `d` 次元の直方体の記述。

`bgeot::convex_product_structure(bgeot::pconvex_structure p1, bgeot::pconv$`
: `p1` と `p2` の直積の記述。

`bgeot::`**`prism_P1_structure`**(dim_type *d*)
: `d` 次元のプリズムの記述。

たとえば、正方形を記述する場合、以下のような表現ができます。

```
p = bgeot::parallelepiped_structure(2);
```

または

```
p = bgeot::convex_product_structure(bgeot::simplex_structure(1),
                                    bgeot::simplex_structure(1));
```

記述子には特に頂点の数 (`nb_points()`) の計算のために面の数 (`p->nb_faces()`) と凸包の次元 (`p->dim()`) が含まれています。他には、各面の頂点の数、面の記述、および（幾何変換の記述に使用される）より基本的な記述への最終的な参照先が情報として含まれます。

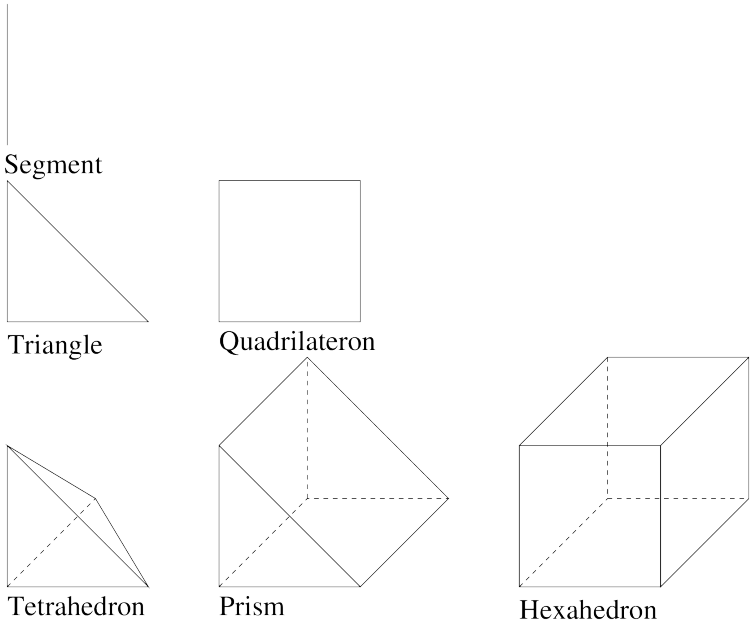


Figure 3.1: 通常の要素

3.2 凸包の参照

参照している凸包は特定の実数要素、すなわち頂点のリストを持つ凸包の構造です。これは有限要素法が定義されている特定の要素を記述しています。ファイル `bgeot_convex_ref.h` 内のオブジェクト `bgeot::convex_of_reference` によって記述されています。ライブラリは、凸包のそれぞれの種類に対して 1 つの記述のみ保持します。したがって、操作するのは記述子上の `bgeot::pconvex_ref` 型のポインタです。

次の関数は記述を構築します。

`bgeot::`**`simplex_of_reference`** (dim_type *d*)
: `d` 次元の参照シンプレックスの記述。

`bgeot::`**`simplex_of_reference`** (dim_type *d*, short_type *k*)
: `k` 次 `d` 次元の Lagrange 格子を参照するシンプレックスの記述。

`bgeot::`**`convex_ref_product`** (pconvex_ref *a*, pconvex_ref *b*)
: 2 つの参照凸包の直積の記述。

`bgeot::`**`parallelepiped_of_reference`** (dim_type *d*)
: `d` 次元を参照する平行六面体の記述。

頂点は、そのような参照要素の古典的な頂点に対応しています。たとえば、三角形の頂点は、$(0,0)$ 、$(1,0)$ 、$(0,1)$ です。これは、図 **通常の要素** に示す構成に対応します。

`p` が `bgeot::pconvex_ref` 型の場合 `p->structure()` は対応する凸包構造です。したがって、例えば `p->structure()->nb_points()` は頂点の数を返します。 `p->points()` 関数は頂点の配列を与え、`p->points()[0]` は最初の頂点です。関数 `p->is_in(const base_node &pt)` は、点 pt が要素内にあ

れば負の実数を返します。関数 `p->is_in_face(short_type f, const base_node &pt)` は点 `pt` が面 `f` にある場合は null を返します。他の関数は `bgeot_convex_ref.h` と `bgeot_convex.h` にあります。

3.3 形状関数の種類

ほとんどの場合、有限要素法の形状関数は、少なくとも参照している凸包においては多項式です。しかし、それ以外の種類の要素を与えることも可能性です。区分多項式、補間曲線ウェーブレットなどの他の種類の基底関数を定義することもできます。

有限要素の記述を使用するためには、形状関数は点で値が設定できなければなりません（`a = F.eval(pt)` ここで `pt` は `base_node` です)。そして、i 次の変数に関する導関数を計算するメソッド（`F.derivative(i)`）を持たなければなりません。

現時点では、ファイル `bgeot_poly.h` と `bgeot_poly_composite.h` では、多項式と区分多項式のみが定義されています。

3.4 幾何変換

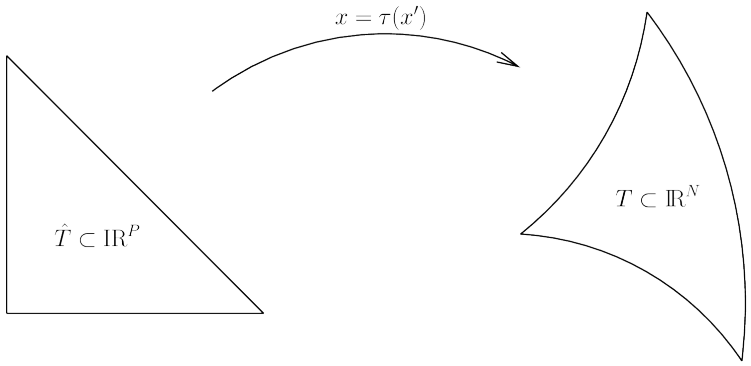


Figure 3.2: 幾何変換

幾何変換は多項式を利用します。

$$\tau : \widehat{T} \subset \mathbb{R}^{\mathrm{P}} \longrightarrow \mathrm{T} \subset \mathbb{R}^{\mathrm{N}},$$

これは、参照要素 $\widehat{T}$ を実数要素 T にマップします。幾何節点は次のように表記されます。

$$g^i, i = 0, \ldots, n_g - 1.$$

幾何変換は n_g 成分多項式ベクトルにより記述されています。 (実際には、非多項式幾何変換を *GetFEM++* の拡張でサポートすることはできますが、これが使用されるのは非常にまれです。)

$$\mathcal{N}(\widehat{x}),$$

ただし

$$\tau(\widehat{x}) = \sum_{i=0}^{n_g-1} \mathcal{N}_i(\widehat{x}) g^i.$$

ここで

$$G = (g^0; g^1; ...; g^{n_g-1}),$$

は $N \times n_g$ の行列でありすべての幾何節点を含みます, ゆえに次式が成り立ちます。

$$\boxed{\tau(\widehat{x}) = G \cdot \mathcal{N}(\widehat{x}).}$$

τ の微分は次式で表されます。

$$\boxed{K(\widehat{x}) := \nabla\tau(\widehat{x}) = G \cdot \nabla\mathcal{N}(\widehat{x}),}$$

ここで $K(\widehat{x}) = \nabla\tau(\widehat{x})$ は $N \times P$ 行列で $\nabla\mathcal{N}(\widehat{x})$ は $n_g \times P$ 行列です。行列 $B(\widehat{x})$ は $N \times P$ の行列であり、$\nabla\tau(\widehat{x})$ の (転置) 擬似逆行列です。

$$\boxed{B(\widehat{x}) := K(\widehat{x})(K(\widehat{x})^T K(\widehat{x}))^{-1},}$$

もちろん $P = N$ の場合、次の式が成り立ちます $B(\widehat{x}) = K(\widehat{x})^{-T}$ 。

幾何変換の記述子に対するポインタは、ファイル `bgeot_geometric_trans.h` で定義されている次の関数によって取得できます。

```
bgeot::pgeometric_trans pgt = bgeot::geometric_trans_descriptor("name of trans");
```

ここで、 `"name of trans"` は次のリストの中から選択します。

- `"GT_PK(n,k)"`

  `n` 次元 `k` 次のシンプレックス変換の記述子 (ほとんどの場合、次数 1 が使用されます)。

- `"GT_QK(n,k)"`

  `n` 次元および次数 `k` の直方体変換の記述子。

- `"GT_PRISM(n,k)"`

  `n` 次元および次数 `k` のプリズム変換の記述子。

- `"GT_PRODUCT(a,b)"`

  2 つの変換 `a` と `b` の直積の記述子。

- `"GT_LINEAR_PRODUCT(a,b)"`

  線形変換（これは前の関数による制約です）である `a` と `b` の 2 つの直積の記述子。これにより、たとえば、平行四辺形を含む通常のメッシュ上で正確な積分を使用することができます。

3.5 有限要素法の記述

有限要素法は節点 a^i と対応する基底関数の集合 n_d により参照要素 $\widehat{T} \subset \mathbb{R}^{\mathrm{P}}$ に対して定義されます。

$$(\widehat{\varphi})^i : \widehat{T} \subset \mathbb{R}^{\mathrm{P}} \longrightarrow \mathbb{R}^{\mathrm{Q}}$$

ここで

$$\psi^i(x) = (\widehat{\varphi})^i(\widehat{x}) = (\widehat{\varphi})^i(\tau^{-1}(x)),$$

補足する線形変換は実基底関数の場合に成り立ちます。

$$\varphi^i(x) = \sum_{j=0}^{n_d-1} M_{ij}\psi^j(x),$$

ここで M は幾何変換が可能な (すなわち、実数要素の) $n_d \times n_d$ 行列です。基本的な Lagrange 要素では、この行列は恒等行列です (単純に無視されます)。この場合、要素は次のように τ -等価になります。

この方法では非線形変換 (主に曲線境界の場合) を使用しても、Hermite 要素 (たとえば Argyris) を汎用的な方法で定義することができます。 $[\widehat{\varphi}(\widehat{x})]$ は i 番目の行が $(\widehat{\varphi})^i(\widehat{x})$ である $n_d \times Q$ の行列です。この表現を使用して、関数は次のように定義されます。

$$f(x) = \sum_{i=0}^{n_d-1} \alpha_i \varphi^i(x),$$

ゆえに

$$\boxed{f(\tau(\widehat{x})) = \alpha^T M[\widehat{\varphi}(\widehat{x})],}$$

ここで α は、 i 番目の成分 α_i を持つベクトルです。

特定の古典的な有限要素法の記述子はファイル `getfem_fem.h` で定義されています。利用可能な有限要素法全体のリストは *ud-appendixa* を参照してください。

有限要素記述子の手法へのポインタは、次の関数を使用することで取得できます。

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

新しい有限要素法を定義する方法は `getfem_fem.cc` を参照してください。

Chapter 4

ライブラリの個別の部分の説明

図 *GetFEM++* **ライブラリのダイヤグラム** は、 *GetFEM++* ライブラリのそれぞれのモジュールのダイヤグラムを説明しています。各モジュールの現在の状態と今後の展望は、 **ライブラリの個別の部分の説明** の節で説明します。

4.1 Gmm ライブラリ

4.1.1 説明

Gmm++ は *GetFEM++* の線形代数と既存のフリーの線形代数ライブラリ (もともとは MTL、Superlu、Blas、Lapack) との間のインターフェイスを作るために設計されていた線形代数のテンプレートライブラリです。独自のベクトルと行列型を持つ独立した自己一貫性のあるライブラリとして進化しています。このライブラリは、他のいくつかのプロジェクトで基礎となる線形代数ライブラリとして使用されるようになりました。

しかし、このライブラリは特定のパッケージのインターフェースとすることも可能です。最小限の互換性を持つベクトルまたは行列型で、 `linalg_traits` 構造で記述された *Gmm++* の汎用アルゴリズムを使用できます。

Gmm++ スタンドアロンバージョンは、 *GetFEM++* のリリース 1.5 以降に配布されています。しかし、リリース 3.0 以降は、完全に *GetFEM++* ファイルから独立しつつも、 *GetFEM++* プロジェクト内部で開発されています。

線形代数の処理に加えて、このライブラリは次のユーティリティを *GetFEM++* に提供しています。

- いくつかの最終的な互換性の問題の修正が `gmm_std.h` で行われます。
- エラー、警告、およびトレース管理が `gmm_except.h` で行われます。
- いくつかの拡張された数学の定義が `gmm_def.h` で行われます。

詳細については *gmm* のドキュメントを参照してください。

4.1.2 ファイル

src/gmm のすべてのファイル

4.1.3 状態

今のところ *Gmm++* は基本的な線形代数について *GetFEM++* のニーズをカバーしています。

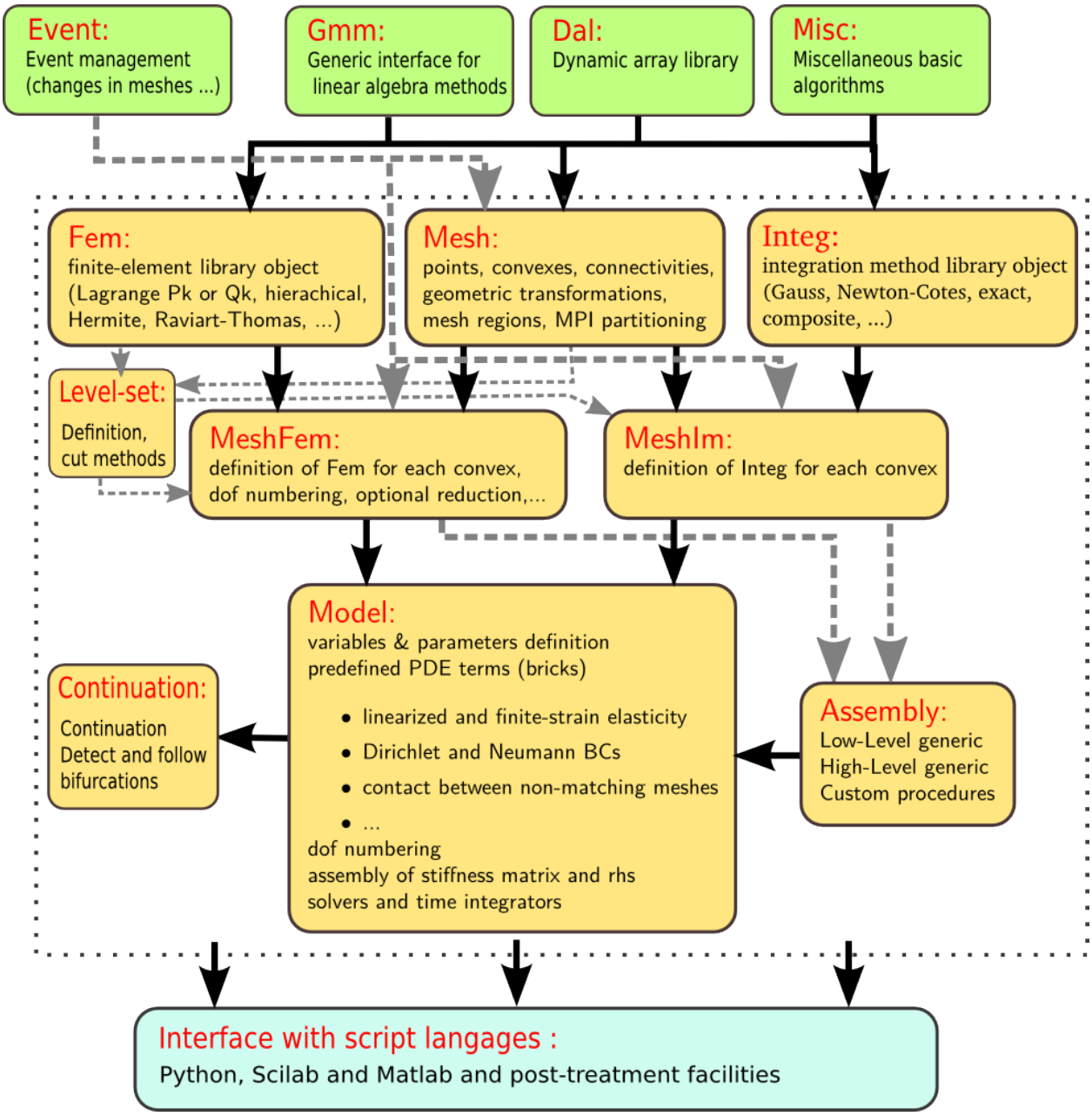


Figure 4.1: *GetFEM++* ライブラリのダイヤグラム

4.1.4 展望

Gmm++ では改善すべき点がいくつかあります。（いくつかの基本型の行列とベクトルの式テンプレートの部分的な導入、疎な部分ベクトルと部分行列をよりコヒーレントな方法で表現する方法、C++の概念の導入など)。ただし、 *Gmm++* は *GetFEM++* のニーズをおおむねカバーしており、線形代数のための C++の参照ライブラリを構築する Glas のような他のプロジェクトがあるので、大規模な変更は必要なさそうです。この部分の開発は安定しています。

現在の *Gmm++* の使命は、プロジェクト全体の新しいニーズをカバーするために、他のいくつかの汎用アルゴリズムのパッケージ (たとえば DIFFPACK) のインターフェイスを揃え続けることです。ライブラリは、現在頻繁に個別のパッケージとして使用されており、いくつかの改善、新しいアルゴリズムや新しいインターフェイスを提案するすべての人の貢献を集約する使命を持っています。

4.2 Dal ライブラリ

4.2.1 説明

GetFEM++ の最初期 (最初のファイルは 1995 に書かれていました) には S.T.L. は使用できず、名前空間 `dal` で定義されているコンテナが使用されていました。現在、 *GetFEM++* では、S.T.L. コンテナが主に使用されています。 `dal` コンテナを使用する理由は、歴史的なものまたはこのコンテナの特有の仕様によるものです。しかし、 *GetFEM++* が新しいコンセプトのコンテナを開発することが目的のプロジェクトではないことは明らかです。したがって、 `dal` コンテナの使用は可能な限りなくす方針です。

さらに、 `dal` には静的に格納された (有限要素法の記述子、補助計算のための中間構造などの) オブジェクトを扱うための基本的なアルゴリズムがいくつか含まれています。

4.2.2 ファイル

ファイル	説明
`dal_config.h`	主に *Gmm++* のヘッダーファイルとしてロードされます
`dal_basic.h`	可変サイズの配列コンテナ、 dal::dynamic_array<T> 。
`dal_bit_vector.h` と `dal_bit_vector.cc`	dal::dynamic_array<T> に基づく改良型ビットベクトルコンテナ。
`dal_tas.h`	dal::dynamic_array<T> に基づくヒープコンテナ。
`dal_tree_sorted.h`	dal::dynamic_array<T> に基づく平衡二分探索木に格納された配列。
`dal_static_stored_objects.h` と `dal_static_stored_objects.cc`	いくつかのオブジェクトとの依存関係を格納することができます。 *GetFEM++* で (有限要素法, 積分法, プリ処理など) を保存するために使用します。
`dal_naming_system.h`	手法記述子に名前を関連付け手法記述子を格納する汎用オブジェクト。有限要素法、積分法および幾何学的変換に使用されます。 dal::static_stored_object を使用しています。
`dal_shared_ptr.h`	boost::shared_ptr を簡略化したバージョンです。
`dal_singleton.h` と `dal_singleton.cc`	OpenMP に対してスレッドセーフに作られた単純な Singleton 実装 (Singleton では、各スレッドが n 個複製されます)。
`dal_backtrace.h` と `dal_backtrace.cc`	デバッグのために、 glibc の backtrace をダンプします。

4.2.3 状態

安定しており、あまり開発されていません。

4.2.4 展望

計画はありません。

4.3 雑多なアルゴリズム

4.3.1 説明

GetFEM++ で使用される雑多な基本的なアルゴリズムと定義の集合。

4.3.2 ファイル

ファイル	説明
`bgeot_comma_init.h`	boost の init.hpp で値のリストのコンテナを初期化することができます。
`bgeot_ftool.h` と `bgeot_ftool.cc`	Matlab ライクな構文のパラメータファイルを読み取ることができる小言語です。構造化メッシュにも使用されます。
`bgeot_kdtree.h` と `bgeot_kdtree.cc`	平衡 N 次元木。点のリストを保存し、与えられたボックス内の点のクイックサーチを行います。
`bgeot_rtree.h` と `bgeot_rtree.cc`	四辺形のツリー。N 次元の四辺形のリストを格納し、指定された点を含む長方形の Quick Search を可能にします。
`permutations.h`	順列の反復処理を可能にします。 `getfem_integration.cc` でのみ使用されます。
`bgeot_small_vector.h` と `bgeot_small_vector.cc`	主にメッシュの節点に使用される低次のベクトルを定義します。操作を最適化しています。
`bgeot_tensor.h`	任意次数のテンソル。アセンブリで使用されます。
`bgeot_sparse_tensors.h` と `bgeot_sparse_tensors.cc`	任意次数の疎テンソル。低水準の汎用アセンブリで使用されます。
`getfem_omp.h` と `getfem_omp.cc`	マルチスレッド、OpenMP および Boost ベースの並列化のためのツール。
`getfem_export.h` と `getfem_export.cc`	pos と vtk 形式でのエクスポート
`getfem_superlu.h` と `getfem_superlu.cc`	(付属のバージョンまたは外部のバージョンの) Superlu のインタフェース

4.3.3 状態

4.3.4 展望

4.4 イベント管理

4.4.1 説明

`mesh`, *mesh_fem*, *mesh_im* と *model* は相互に依存関係があります。たとえば要素がメッシュに対応していない場合、 *mesh_fem* オブジェクトは反応する必要があります。

4.4.2 ファイル

ファイル	説明
getfem_context.h と getfem_context.cc	イベントを管理するすべてのオブジェクトの派生元となるクラス *context_dependencies* を定義します。

4.4.3 状態

オブジェクトの単純な依存性に対処するための主なツールは getfem_context.h にあります。オブジェクト context_dependencies はここで定義されています。オブジェクトの依存関係を処理するために、オブジェクト context_dependencies はオブジェクトの親クラスである必要があります。オブジェクトには次のメソッドが追加されています。

add_dependency(ct)
: オブジェクト (親クラスとして context_dependencies を持つ必要があります) を、現在のオブジェクトが依存するオブジェクトのリストに追加します。

touch()
: オブジェクトに何か変化があったことを従属オブジェクトに知らせます。

context_check()
: オブジェクトを更新するかどうかを確認します。更新する場合は、最初に依存関係リストにチェックを行い、オブジェクトの更新関数を呼び出します。(依存関係の更新関数は、現在のオブジェクトの更新関数の前に呼び出されます)。

context_valid()
: オブジェクトがまだ有効なコンテキストを持っているかどうか、つまり、依存関係リスト内のオブジェクトがまだ存在するかを表示します。

さらに、オブジェクトは次のメソッドを定義する必要があります。

```
``void update_from_context(void) const``
```

このメソッドはコンテキストが変更された際に context_check() の後に呼び出されます。

オブジェクト *mesh* には追加のシステムが提供されています。 *mesh_fem* と *mesh_im* の 2 つの呼び出しの間で変更された要素を検出するために個々の要素はオブジェクトのバージョン番号を所持しています。

4.4.4 展望

いくつかのオブジェクトのイベント管理は、注意して分析する必要があります。 *mesh_level_set* 、 *mesh_fem_level_set* 、 *partial_mesh_fem* などのインスタンスの場合です。

完全な挙動をするイベント管理システムにするためにはまだ改善の余地があります。

4.5 Mesh モジュール

4.5.1 説明

ライブラリのこの部分ではメッシュの格納および管理を行います 。 (実際の) 要素は、それぞれの面で相互に接続されている必要があります。そのために、要素の概念、参照要素、メッシュの構造、節点の集合、幾何変換、メッシュの境界の一部さらにサブ領域を開発しています。

GetFEM++ には現時点で実用的なメッシュ機能はありません。複雑な対象のメッシュは Gmsh または GiD などの既存のメッシャーからインポートする必要があります。いくつかのメッシュインポート関数が実装されており、他のフォーマットに対しても簡単に拡張することができます。

`getfem_mesh.h` ファイル内で宣言されているメッシュを表すオブジェクトで、 *GetFEM++* 内でのメッシュの処理の基礎としてメッシュに依存する構造体 (MESHFEM および MESHIM モジュールを参照) を管理し、(要素の追加または除去などの) メッシュ修正にも対応します。

4.5.2 ファイル

ファイル	説明
`bgeot_convex_structure.h` と `bgeot_convex_structure.cc`	頂点の座標を含まない要素の構造について記述します。
`bgeot_mesh_structure.h` と `bgeot_mesh_structure.cc`	節点の座標を含まないメッシュの構造について記述します。
`bgeot_node_tab.h` と `bgeot_node_tab.cc`	節点と複数の格納節点の高速検索を可能にし、近すぎる節点を識別する節点コンテナ。
`bgeot_convex.h`	要素とその頂点について記述します。
`bgeot_convex_ref.h` と `bgeot_convex_ref.cc` と `bgeot_convex_structure.cc`	参照要素について記述します。
`bgeot_mesh.h`	節点の集合に対するメッシュを記述します (ただし、幾何変換については記述しません)。
`getfem_mesh_region.h` と `getfem_mesh_region.cc`	メッシュ領域 (メッシュの境界または一部) を表すオブジェクト。
`bgeot_geometric_trans.h` と `bgeot_geometric_trans.cc`	幾何変換について記述します。
`bgeot_geotrans_inv.h` と `bgeot_geotrans_inv.cc`	幾何変換の反転ツール。
`getfem_mesh.h` と `getfem_mesh.cc`	メッシュ (幾何学的変換、部分メッシュ、並列化のサポート) を全て記述します。メッシュ洗練のための Bank のアルゴリズムが含まれます。
`getfem_deformable_mesh.h`	変位に対応してメッシュを変形し、復元もできるオブジェクトを定義します。
`getfem_mesher.h` と `getfem_mesher.cc`	任意の次元の実験的メッシャー。（ノード最適化をしているため）非常に遅くそして注意して使用する必要があります。いくつかのレベル集合によって定義されたジオメトリをメッシュ分割します。
`getfem_import.h` と `getfem_import.cc`	さまざまな形式のメッシュファイルをインポートします
`getfem_regular_meshes.h` と `getfem_regular_meshes.cc`	構造化メッシュを生成します
`getfem_mesh_slicers.h` と `getfem_mesh_slicers.cc`	スライスはいくつかのスライス操作 (メッシュを平面で切る、球体と交差させる、境界面を取るなど) を行ってメッシュから構築します。(VTK または OpenDX への結果の出力等の) 後処理で使用します。
`getfem_mesh_slice.h` と `getfem_mesh_slice.cc`	メッシュスライスを格納します。

4.5.3 状態

安定しており、あまり進化していません。

4.5.4 展望

現時点では、このモジュールは 2 つの部分に分割されており、異なる名前空間に分かれています。もちろん、1 つの名前空間 (getfem) でモジュールを生成する方が首尾一貫しています。

ノート: ファイル名 bgeot_mesh.h を getfem_basic_mesh.h に変えることができます。

メッシュの効率的な保存と（節点、要素の追加、面の削除、隣接要素や孤立した面の検索などの）操作の実装方法に関する参考文献をレビューすればさらにメッシュ構造を進化させることができると思われます。

(bgeot_node_tab.cc 内の) 感知アルゴリズムは近すぎる節点を識別するためのものです。より多くの調査 (およびドキュメンテーション) が必要と思われます。

4.6 Fem モジュール

4.6.1 説明

GetFEM++ の Fem モジュールは要素レベル有限要素と自由度を記述します。有限要素法はさまざまな種類を使用することが可能です。幾何変換のあるなしか、等価なスカラーかベクトルか、多項式、区分多項式または非多項式かなどです。また、自由度は一般的に 2 つの隣接要素間で互換性のある自由度となるように記述する必要があります (例えば Lagrange 2 次要素を別の Lagrange 1 次要素に接続する場合)。

4.6.2 ファイル

ファイル	説明
bgeot_poly.h と bgeot_poly_composite.h と bgeot_poly.cc と bgeot_poly_composite.cc	いくつかのクラスでは、参照要素の有限要素法の形状関数を記述するために、多項式と区分的多項式を表現します。
getfem_fem.h と getfem_fem.cc と getfem_fem_composite.cc	有限要素と自由度の記述子。 getfem_fem.cc で定義される多項式有限要素と getfem_fem_composite.cc の区分多項式有限要素。
getfem_fem_global_function.h と getfem_fem_global_function.cc	ユーザーが指定したグローバル関数として定義された基底関数を持つ有限要素法を定義します。特異関数による強化やメッシュレス法の実装に有用です。
getfem_projected_fem.h と getfem_projected_fem.cc	(mesh_fem で表される) 別のメッシュ上の有限要素空間の射影である有限要素法を定義します。高水準汎用構築言語では、補間変換でもこの機能が提供されます。
getfem_interpolated_fem.h と getfem_interpolated_fem.cc	別のメッシュ上の有限要素空間 (mesh_fem で表される) の補間である有限要素法を定義します。高水準汎用構築言語では、補間変換によってこの機能も提供されます。

4.6.3 状態

2 つのファイル getfem_fem.cc と getfem_fem_composite.cc には主な基本的有限要素要素のすべての記述が含まれています。定義された有限要素の詳細なリストは *ud-appendixa* を参照してください。

他には getfem_fem_level_set.h ファイルのように複数の有限要素を定義する複雑な構造は 1 つまたは複数のレベル集合で要素を “切断” することが可能です。

自由度を記述する方法は、おおむね必要事項 (汎用的な方法で別の要素から接続された自由度) を満たしますが、少しあいまいで、あまりにも複雑です。

逆に、補助行列 M で非等価要素を表現する方法は、いくつかの要素 (Hermites 要素、 Argyris など) でその効率性が証明されています。

4.6.4 展望

このモジュールの主な不満は、自由度を完全に記述できていないということです。自由度の記述が一時的な設計であるために、全ての要素を構築する方法についてのドキュメンテーションを 1 つにまとめられていません。このモジュールを安定させるためには自由度の記述に関するの設計を完全なものにしなければなりません。また、立体求積法と同様の方法で部分的に外部化できる有限要素の記述法は最も単純な有限要素 (等価および多項式有限要素) のみと思われます。

4.7 Integ モジュール

4.7.1 説明

立体求積法モジュールでは、参照要素の数値積分法へのアクセスが可能です。実際には、いくつかの完全積分法へのアクセスが必要なため、いくつかの立体求積法式が含まれていません。完全積分法は多項式要素と affine 幾何変換に対してのみ使用できます。これが完全積分法が広く使用されない理由です。立体求積法式の記述は、`getfem_integration.h` または *GetFEM++* のディレクトリ `cubature` の説明ファイルを介してファイル内で直接実行されます。新しい立体求積法式の追加は非常に簡単であり、すでに定義されている要素を参照し、ファイル内の Gauss 点のリストをこのディレクトリに追加するだけです。さらに、領域の境界に定義された項を積分するために、記述には要素の各面における同じ次数の手法への参照も含める必要があります。

4.7.2 ファイル

ファイル	説明
`getfem_integration.h` と `getfem_integration.cc` と `getfem_integration_composite.cc`	積分法の構造、基本的な積分法、積分法および複合法の積。
`getfem_im_list.h`	file generated by `cubature/make_getfem_list` with the integration methods defined in cubature directory. This gives the possibility to define a new integration method just listing the Gauss points and weigth in a text file.

4.7.3 状態

このモジュールは、プロジェクトの現在のニーズを満たしており、安定しています。使用可能な立体求積法式のリストは *ud-appendixb* を参照してください。

4.7.4 展望

今のところ変更は必要ありません。新しい立体求積法の式を追加する方法 (記述子ファイル形式) に関するドキュメンテーションを完成させる必要があります。

4.8 MeshFem モジュール

4.8.1 説明

MeshFem モジュールは、特定のメッシュに対する有限要素法 (空間) を表すことを目的としています。mesh_fem オブジェクトは、与えられたメッシュにリンクされ、メッシュの変化 (特に要素の追加または削除) に対応します。mesh_fem オブジェクトはメッシュの各要素に異なる有限要素メソッドを関連付けることができます。もちろん、すべての要素が同じ有限要素法を使用するのが最も一般的なケースです。

4.8.2 ファイル

ファイル	説明
getfem_mesh_fem.h と getfem_mesh_fem.cc	メッシュ全体の有限要素を表す構造体を定義します。メッシュの各要素は、有限要素法に関連付けられています。これは、グローバルな線形リダクションを可能する非常に複雑な構造です。また、自由度の識別とナンバリングも行います。
getfem_mesh_fem_global_function と getfem_mesh_fem_global_function.cc	fem_global_function として定義された有限要素法で mesh_fem を定義します。リンクされた fem_global_function の基底関数のリストを更新するための便利なメソッドを提供します。
getfem_mesh_fem_product.h と getfem_mesh_fem_product.cc	2 つの有限要素法の直積である mesh_fem オブジェクトを生成します。 Xfem に役立ちます。
getfem_mesh_fem_sum.h と getfem_mesh_fem_sum.cc	2 つの有限要素法の直和である mesh_fem オブジェクトを生成します。 Xfem に役立ちます。
getfem_partial_mesh_fem.h と getfem_partial_mesh_fem.cc	自由度数を減らした mesh_fem を生成します。
getfem_interpolation.h と getfem_interpolation.cc	異なるメッシュ間の 2 つの有限要素法の間の補間。高水準汎用構築の補間機能を代わりに使うことができます。
getfem_derivatives.h	(不連続) Lagrange 有限要素における有限要素場の微分の補間の代わりに、高水準の汎用構築の補間機能を使用することができます。
getfem_inter_element.h と getfem_inter_element.cc	要素間の計算のためのフレームワークを作る試み (例えば法線微分における分岐)。開発は継続中で、汎用構築言語に統合されるかもしれません。
getfem_error_estimate.h と getfem_error_estimate.cc	誤差推定の計算のためのフレームワークを作る試みです。継続中であり、汎用構築言語に統合されるかもしれません。
getfem_crack_sif.h	亀裂の SIF (応力拡大係数) 計算のためのサポート関数。
getfem_torus.h と getfem_torus.cc	半径の次元で 2 次元構造を拡張した mesh_fem オブジェクトを作成します。

4.8.3 状態

安定しており。あまり開発されていません。

4.8.4 展望

自由度のナンバリングの並列化が行われています。最適 (単純な) アルゴリズムが存在します。

4.9 MeshIm モジュール

4.9.1 説明

メッシュ全体に積分法を定義します。

4.9.2 ファイル

ファイル	説明
`getfem_mesh_im.h` と `getfem_mesh_im.cc`	メッシュの各要素に対して積分法を定義するオブジェクト。従属しているメッシュの変更 (要素の追加または削除) に対応します。
`getfem_im_data.h` と `getfem_im_data.cc`	mesh_im オブジェクトの各 Gauss 点のスカラー、ベクトルまたはテンソルを表すオブジェクトを定義します。例えば塑性近似で使用されます。弱形式言語により、任意の式による補間を行うことができます。

4.9.3 状態

安定しており、それほど進化していません。

4.9.4 展望

4.10 Level-set モジュール

4.10.1 説明

1 つまたは複数のレベル集合に関して、level-set オブジェクトと切断メッシュ、積分法、有限要素法を定義します。

4.10.2 ファイル

ファイル	説明
getfem_level_set.h と getfem_level_set.cc	オプションの補助的なレベル集合関数を使用して、 (Lagrange 有限要素法で定義されたスカラー場での) レベル集合関数を定義します。
getfem_mesh_level_set.h と getfem_mesh_level_set.cc	1 つまたは複数のレベル集合に対してメッシュを切断します。
getfem_fem_level_set.h と getfem_fem_level_set.cc	要素に依存し、1 つまたは複数のレベル集合によって切断される特殊な有限要素法を定義します。
getfem_mesh_fem_level_set.h と getfem_mesh_fem_level_set.cc	1 つまたは複数のレベル集合によって切断された形状関数を持つ mesh_fem オブジェクトを生成します。
getfem_mesh_im_level_set.h と getfem_mesh_im_level_set.cc	レベル集合によって切断された積分法を表す mesh_im を生成し、レベル集合もしくはその他の側面、レベル集合自体の両側または片側の側面となります。
getfem_level_set_contact.h と getfem_level_set_contact.cc	埋め込みの位置決めを簡単に解析するための、レベル集合ベースの有限滑り接触アルゴリズム。
getfem_convect.h	ベクトル領域に対応する対流の量を計算します。たとえば、レベル集合関数を開発し計算するために使用されます。Galerkin 特性法。

4.10.3 状態

安定しています。

4.10.4 展望

異なる領域を計算するアルゴリズムを明確にする必要があります。

4.11 *GetFEM++* の高水準汎用構築モジュール

4.11.1 説明

GetFEM++ の高水準の汎用構築モジュールと弱形式言語は、偏微分方程式の問題の弱定式化を記述することができるキーモジュールです。ユーザードキュメント *ud-gasm-high* の説明を参照してください。

4.11.2 ファイル

ファイル	説明
getfem_generic_assembly.h	エクスポートされた定義のメインヘッダー。汎用構築を使用するには、このヘッダーをインクルードするだけで十分です。モジュールの他のヘッダーは内部で使用されます。
getfem_generic_assembly_tree.h と getfem_generic_assembly_tree.cc	ツリー構造とその基本操作を定義します。アセンブリ文字列を読み取って構文ツリーに変換し、ツリーを文字列にする逆変換を作成します。
getfem_generic_assembly_function_and_operators.h と getfem_generic_assembly_function_and_operators.cc	再定義関数の定義と弱形式言語の非線形演算子。
getfem_generic_assembly_semantic.h と getfem_generic_assembly_semantic.cc	構文ツリーの意味解析と強化。変数に対するツリーの導出や、式の勾配に対応するツリーの計算など、いくつかの操作を含みます。
getfem_generic_assembly_workspace.cc	(getfem_generic_assembly.h で定義されている)workspace オブジェクトのメソッド 。
getfem_generic_assembly_compile_and_exec.h と getfem_generic_assembly_compile_and_exec.cc	最適化された命令の定義、コンパイルによる最適化された命令の作成と Gauss 点/補間点での命令の実行を行います。
getfem_generic_assembly_interpolation.cc	補間演算と補間変換。

4.11.3 いくつかの実装の詳細

アセンブリ文字列は src/getfem_generic_assembly.cc の一連の関数によってアセンブリツリーに変換されます。プロセスには 6 つのステップがあります。

- ga_get_token(...) を使用した字句解析。
- ga_read_string(...) による構文解析と構文木への変換。
- ga_semantic_analysis(...) による定数式の意味解析, 単純化 (事前計算) とツリーの強化。
- ga_derivative(...) によるアセンブリツリーの (自動) 微分の表現
- ga_gradient(...) によるアセンブリツリーの (自動) 勾配計算の表現
- ga_compile(...) による最適化された一連の命令のコンパイル。
- ga_exec(...) による命令とアセンブリの直列実行。

これらの手順は、アセンブリのはじめに 1 回だけ実行されます。最終的なツリーは、各要素の各 Gauss 点で実行される最適化された命令でコンパイルします。コンパイルではいくつかの最適化が実行されます。繰り返しの項は自動的に検出され、1 回だけ実行されます。メッシュの簡略化はベクトル有限要素法の要素が均一な型を持っている場合に行われます。

さらに、補間操作 (ga_interpolation(...), ga_interpolation_exec(...), ga_interpolation_Lagrange_fem, ga_interpolation_mti, ga_interpolation_im_data, ...) の具体的な関数を実行します。

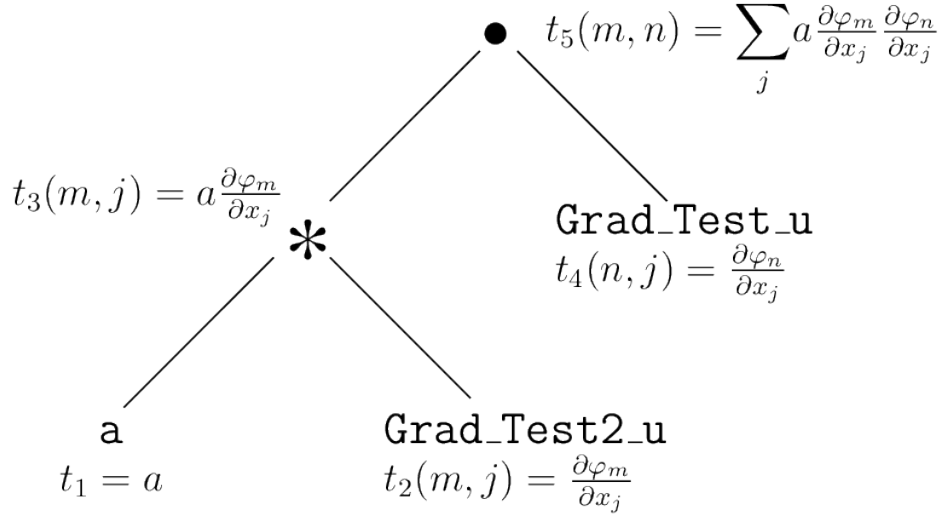
4.11.4 アセンブリツリー

アセンブリ文字列は ga_read_string(...) でアセンブリツリーに変換されます。アセンブリツリーは、各ステップ (意味分析、導出、コンパイル) で徐々に強化される構文ツリーです。

オブジェクト ga_tree はアセンブリツリーを表します。これはツリーのルートへのポインターのみを含むコピー可能オブジェクトです。各ツリーのノードは、主に次の情報を含むオブジェクト ga_tree_node です。

- node_type (関数, 変数値, 勾配値, 操作 ...)
- 節点操作の操作の種類。
- アセンブリテンソル: 中間結果を計算するために最適化された命令によって実行時に使用されます。最終的な結果は、実行の終了時にルートノードの (各 Gauss 点での) アセンブリ文字列で使用されます。
- 項の種類: 値、1 次の項 (i 番目の 1 次の試行関数)、2 次の項 (2 次の試行関数) または 1 次と 2 次の試行関数 (接線項)。
- 1 次または 2 次の項の試行関数の変数名。
- 親ノードへのポインター。
- 子ノードへのポインター。

たとえば、Laplacian 問題の剛性マトリックスのアセンブリ文字列 “a*Grad_Test2_u.Grad_Test_u” のアセンブリツリーは、各ノードのアセンブリテンソルをもとに次のように表現できます。



4.11.5 アセンブリテンソル

アセンブリテンソルは、各ノードで `bgeot::tensor<double>` オブジェクトによって表されています。ただし、アセンブリテンソルには複数の形式があり、`src/getfem_generic_assembly.cc` にはアセンブリテンソルのための詳細構造を実装しています。

- 通常のテンソル。1 番目と 2 番目のインデックスは、ノードが 1 次または 2 次の項を表す場合、試行関数のローカルインデックスを表します。 *GetFEM++* ではすべてのテンソルは Fortran の次数で保存されます。これは例えば $N \times P \times Q$ であるテンソルは `t(i, j, k) = t[i + j*N + k*N*P]` となるということです。
- コピーされたテンソル。ノードが、既にコンパイルされたものと比較してまったく同じ式を持つ場合、アセンブリテンソルには、既にコンパイル済みのノードのアセンブリテンソルへのポインタが設定されます。結果として、不要なコピーは作成されません。
- スパースにリストされた疎テンソル。ベクトルフィールドを使用する場合、有限要素法は各成分に適用されます。その結果、ベクトの基底関数が 0 以外の成分を 1 つだけ持ち、一部の成分が複製されます。テンソルは、その種の小さな疎テンソル形式で効率化のために複製されるので、完全に表現されます。ただし、一部の操作は、特定のスパース (および複製) の知見を使用して最適化できます。これにより、縮約の複雑さの度合いを変更することができます。効率的な使用を可能にするためにテンソルはいくつか

のスパースフォーマットでラベル付けがされています (ベクトルテンソルで適用される操作から来るベクトル化とフォーマット)。これにより、以下に示すようなスパース形式が得られます。

- 1：ベクトル化ベースのスパース形式：テンソルはベクトル化値を表します。 縮約テンソルの各値は、ベクトル化テンソルの *Q* 成分に対して繰り返されます。メッシュ寸法は *N* で示されます。たとえば、 φ_i が要素 M 上のローカル基底関数で評価が Gauss 点 x の場合、ベクトル化されていないテンソルは $\bar{t}(i) = \varphi_i(x)$ であり、ベクトル化されたものは $t(j,k) = \varphi_{j/Q}(x)\delta_{k,j \bmod Q}$ となります。ここで j/M は整数の除算です。 $M = 2$ 、 $Q = 2$ 、$N = 3$ の場合、2 つのテンソルの成分は次の表のように表されます。

スカラテンソル	ベクトルテンソル
$\bar{t}(i) = \varphi_i(x)$ $[\varphi_0(x), \varphi_1(x)]$	$t(j,k) = \varphi_{j/Q}(x)\delta_{k,(j \bmod Q)}$ $[\varphi_0(x), 0, \varphi_1(x), 0, 0, \varphi_0(x), 0, \varphi_1(x)]$

- 2: Grad の縮約形式

スカラテンソル	ベクトルテンソル
$\bar{t}(i,j) = \partial_j\varphi_i(x)$ $[\partial_0\varphi_0(x), \partial_0\varphi_1(x), \partial_1\varphi_0(x), \partial_1\varphi_1(x), \partial_2\varphi_0(x), \partial_2\varphi_1(x)]$	$t(k,l,m) = \partial_m\varphi_{k/Q}(x)\delta_{l,(m \bmod Q)}$

- 3: Hessian 縮約形式
- 10: ベクトル化された質量: テンソルは 2 つのベクトル化された基底関数のスカラー積を表します。つまり、 $k \neq l$ と $t(i*Q+k, j*Q+k)$ に対して $t(i*Q+k, j*Q+l) = 0$ であるテンソル $t(\cdot,\cdot)$ は $0 \leq k < Q$ と等価です。

4.11.6 最適化された命令

変数の評価、操作、ベクトルと行列のアセンブリのための最適化された命令など。説明を準備中。

4.11.7 定義済み関数

ライブラリ *GetFEM++* の弱形式言語は弱定式化を記述するために (または基本的な代数計算もするため) いくつかの定義済みス

- (複数の) 引数に指定された値を計算する C++ の関数。
- (可能な限り無限の) 範囲で各第 1 引数の関数をサポートします (これは表現の簡素化のためです)。
- 既知の関数の導関数に対応する文字列

新しい定義済み関数は簡単に追加できます。ファイル `src/getfem_generic_assembly.cc` の init_predefined_functions() を参照してください。微分を与える方法についての説明を準備中。

4.11.8 定義済み非線形演算子

説明を準備中。

4.11.9 状態

安定しています。

4.11.10 展望

- 複雑なデータへの拡張は可能でしょうか?
- さらなる簡素化: 処理数を減らすために、いくつかの項 (例えばスカラー項) を自動的に因数分解するとよいかもしれません。

4.12 *GetFEM++* の低水準汎用構築モジュール

4.12.1 説明

汎用構築の最初のバージョンです。テンソルの低減の基礎となる部分です。非線形項に使用するにはあまり便利ではありません。高水準の汎用構築が推奨されるようになりました。

4.12.2 ファイル

ファイル	説明
`getfem_mat_elem_type.h` と `getfem_mat_elem_type.cc`	要素行列の成分の基礎型を定義します。
`getfem_mat_elem.h` と `getfem_mat_elem.cc`	要素行列の計算について記述します。
`getfem_assembling_tensors.h` と `getfem_assembling_tensors.cc`	アセンブリを実行します。
`getfem_assembling.h`	様々なアセンブリの項 (線形弾性, 汎用楕円項, Dirichlet 条件...

4.12.3 状態

安定しています。

4.12.4 展望

現在、高水準汎用構築の開発に注力しているため、進んでいません。

4.13 Model モジュール

4.13.1 説明

モデル (変数、データ、および変数をリンクする方程式の項) について記述します。

4.13.2 ファイル

ファイル	説明
getfem_models.h と getfem_models.cc	モデルオブジェクト、その内部と標準ブリック (線形弾性、汎用ひずみブリック、Dirichlet 境界条件 ...) を定義します。
getfem_model_solvers.h と getfem_model_solvers.cc	モデルオブジェクトの標準ソルバーを定義します。
getfem_contact_and_friction_common.h と getfem_contact_and_friction_common.cc	変形体の接触/摩擦条件のための共通アルゴリズム
getfem_contact_and_friction_integral.h と getfem_contact_and_friction_integral.cc	積分型の微小すべり接触/摩擦ブリック。
getfem_contact_and_friction_large_sliding.h と getfem_contact_and_friction_large_sliding.cc	有限滑り接触/摩擦ブリック。
getfem_contact_and_friction_nodal.h と getfem_contact_and_friction_nodal.cc	微小すべり nodal 接触/摩擦ブリック。
getfem_Navier_Stokes.h	Navier-Stokes ブリックに取り組んでいます。改善の余地があります。
getfem_fourth_order.h と getfem_fourth_order.cc	Bilaplacian と Kirchhoff-Love 板ブリック
getfem_linearized_plates.h と getfem_linearized_plates.cc	Mindlin-Reissner 板ブリック
getfem_nonlinear_elasticity.h と getfem_nonlinear_elasticity.cc	有限変形弾性ブリック。
getfem_plasticity.h と getfem_plasticity.cc	塑性ブリック。

4.13.3 状態

新しいモデルが追加されていきます。

4.13.4 展望

より多くの板、荷重およびシェルブリック、塑性大変形など。

4.14 Continuation モジュール

4.14.1 説明

パラメータ (連続法) に関して解を追跡し、分岐の検出と追跡を行います。低い正則問題 (Lipschitz 正則) で実行されます。適合した Moore-Penrose の継続法を使用します。

4.14.2 ファイル

ファイル	説明
getfem_continuation.h と getfem_continuation.cc	汎用的な連続および分岐法

4.14.3 状態

すでに汎用的で高度な機能があります。

4.14.4 展望

まだ開発中です。

4.15 スクリプト言語 (Python、Scilab、Matlab) のインタフェース

GetFEM++ の簡略化された (しかしかなり完全な) インターフェイスが提供されています。これによりいくつかのスクリプト言語で getfem を使用することができます。

4.15.1 説明

すべてのソースは `interface/src` ディレクトリにあります。インターフェイスは、1 つの大規模なライブラリ `getfemint` (getfem interaction の略) で構成され、 *GetFEM++* ライブラリ上のレイヤーとして機能します。python 、 matlab と scilab のインターフェイスに使用されています。

(swig などのツールを使用することも可能ですが、) このインターフェイスは、c++ ソースから自動的に生成されているものではありません。このインターフェースは getfem のために簡素化され、一貫性のあるインターフェイスとして設計されています。(言語が密な配列操作の構造を提供しているのならば) 新しい言語を追加することはとても簡単です。

4.15.2 ファイル

ディレクトリ `interface/src` 内のすべてのファイル。主なファアイルについて簡単に説明します。

- `getfem_interface.cc` 。

  これは、スクリプト言語と getfem インターフェイスの間の Bridge です。関数 getfem_interface_main は `extern "C"` 関数としてエクスポートされます。これはスクリプト言語と getfem インターフェイスの間の c++ が起こす障害です (C インターフェイスのみをエクスポートすると、多くの複雑な問題が回避されます)。

- `matlab/gfm_mex.c` 。

  matlab インターフェイス。 getfem のインターフェースに関連するファイルは `getfem_interface.h` のみです。

- `python/getfem_python.c` 。

  python インターフェイス。 getfem のインターフェースに関連するファイルは `getfem_interface.h` のみです。

- `gfi_array.h`, `gfi_array.c` 。

  `gfm_mex.c` と `getfem_python.c` はどちらも `getfem_interface_main()` から配列とオブジェクトハンドルを送受信する方法に簡単な規則が必要です。このファイルはそのような機能を提供します。

- `getfemint_gsparse.h`, `getfemint_precond.h`, など。

  必要に応じて、インタフェースオブジェクトに固有のファイルを出力します。(getfemint_gsparse は、異なる種類のストレージの切り替えが可能ないくつかの種類の可変疎行列と複素要素の実数部分をエクスポートします) 。

- gf_workspace.cc, gf_delete.cc 。

  getfem オブジェクトのメモリ管理を行います。たとえば mesh と mesh_fem の間の依存関係を処理するレイヤがあります。これを使用して別の getfem_object がまだある間、そのオブジェクトが破壊されていないことを監視します。その目的は、ユーザーが getfem インターフェイスに不正な引数を渡すことによって getfem (およびホストプログラム、matlab、scilab または python) をクラッシュさせることができないようにすることです。

  また、(matlab は “object destructors” を持っていないので) matlab では多数の getfem オブジェクトのハンドリングとクリーニングを簡素化するように設計された workspace のスタックのようなものを提供しています。

- getfemint.h, getfemint.cc 。

  入力と出力の引数のリストを getfem インターフェイス関数でパースするための mexarg_in, mexarg_out クラスを定義します。 “mex” への参照が gfm_mex.c に移されたので、名前空間はもう必要ありません。

- gf_mesh.cc, gf_mesh_get.cc, gf_mesh_set.cc, gf_fem.cc, など。

  エクスポートされるすべての関数は、オブジェクトの種類 (gf_mesh*, gf_mesh_fem*, gf_fem*) で並べ替えられた getfem インターフェイスになります。オブジェクトのコンストラクト (gf_mesh) 、オブジェクトの変更 (gf_mesh_set) 、オブジェクトの参照 (gf_mesh_get) 。これらの各ファイルには、mexargs_in と mexargs_out の引数スタックを受け取る main 関数が 1 つ含まれています。解析の後、通常最初の引数をサブ関数 (matlab で gf_mesh_get('nbpts') 、または python で Mesh.nbpts()) の名前として解釈します。

- matlab/gfm_rpx_mexint.c 。

  --enable-matlab-rpc が ./configure スクリプトに渡されるときに使用される gfm_mex.c の代替ファイルです。その場合、 matlab インターフェイスは “getfem_server” プログラムでソケットを介して通信するので、そのサーバープログラムをデバッグすることが可能です。メモリリークまたは matlab の何かを混乱させず識別するインターフェイスをデバッグするのがこのファイルの主な用途です (デバッグは苦痛です)。

- python/getfem.py 。

  python インタフェースは python スクリプト “bin/extract_doc.py” によってコンパイル中に生成される “getfem.py” ファイルで利用可能です。

4.15.3 インターフェイスのオブジェクト、メソッド、および関数

インターフェイスによって操作される主なものは、 (Fem, Mesh, MeshFem, Model ... などの) いくつかのオブジェクト、関連するメソッド、およびこれらのオブジェクトで定義されている関数です。

サポートしている各スクリプト言語 (Python、Scilab、Matlab) で別々に対応することなく、インターフェイスに新しいオブジェクト、メソッド、および関数を容易に追加できるようにするために工夫をしています。

さまざまなオブジェクト、メソッド、および関数のインターフェイスを構築するために必要なすべての情報は、ファイル *interface/src/gf*.cc* に含まれています。python スクリプト (*bin/extract_doc*) は、そこから全ての必要な情報を取得しファイルを生成します。生成されるのは python ファイル getfem.py と、 matlab の m-ファイルによる個別の関数やオブジェクト (サブディレクトリを含む) と、自動ドキュメンテーションの出力ファイルです。

すべてを自動で動作させるには、一定のルールを尊重する必要があります。

- オブジェクトはインターフェイス上の 3 つのファイルで定義する必要があります。
  - gf_objectname.cc : オブジェクトのコンストラクタが含まれています。
  - gf_objectname_get.cc : (存在する場合) オブジェクトに関する情報を取得するメソッドが含まれています。
  - gf_objectname_set.cc : (存在する場合) オブジェクトを変換するメソッドが含まれています。

- 関数のリストは全て gf_commandname.cc で定義されています。このファイルにはサブコマンドのリストが含まれています。
- 各ファイルについて、関数またはメソッドのリストに関する主な解説は、タグ '/@*GFDOC*' と '@/' で区切られています。オブジェクトのコンストラクタに対応するファイルの場合、注釈はオブジェクトの説明としてください。
- それぞれの重要なファイル gf_*.cc には、オブジェクトまたはサブコマンドのメソッドを定義できるマクロが含まれています。特に、このシステムは、呼び出されたメソッド/関数の効率的な検索をすることができます。このマクロでは、次の構文を使用して新しいメソッド/関数を宣言できます。

```
/*@GET val = ('method-name', params, ...)
   Documention of the method/function.
@*/
sub_command
("method-name", 0, 0, 0, 1,
  ...
  body of the method/function
  ...
);
```

最初の 3 行は、特別な構文でメソッド/関数の呼び出しを記述し、また、ドキュメントに含まれるメソッド/関数の c++ に関する解説です。これは Python、Matlab と Scilab のための適切なインターフェイスを生成するために使用されるため、この解説の最初の行は重要です。

メソッド/関数の呼び出しの説明の構文は次のとおりです。 /*@ の後には、特別なキーワードを記入する必要があります。これは、INIT, GET, SET, RDATTR または FUNC のいずれかです。キーワード INIT は、これがオブジェクトのコンストラクタの記述であることを意味します。RDATTR は、オブジェクト属性を得る短いメソッドです。 GET は、変更をしないオブジェクトのメソッドです。 SET はオブジェクトを変更するメソッドのためのもので、 FUNC は関数リストのサブコマンドです。

メソッド/関数が値を返す場合は、 = に続いて (任意の) 戻り値の名前が返されます。

メソッド/関数のパラメータについて説明します。メソッドの場合、オブジェクト自体についての言及は行いません。最初のパラメーターはメソッドまたは単一引用符で示すサブコマンド名である必要があります (この名前がドットで始まる特別な場合もあります。これは、コマンド名が不要なメソッド/関数であることを意味します)。

その他のパラメーターがある場合は、型を使用して宣言します。定義済みの型は次のとおりです。

- @CELL : セルの配列、
- @imat : 整数の行列、
- @ivec : 整数のベクトル、
- @cvec : 複素数値のベクトル,
- @dcvec : 複素数値のベクトル、
- @dvec : 実数値のベクトル、
- @vec : 実数または複素数値のベクトル、
- @dmat : 実数値の行列、
- @mat : 実数または複素数値の行列、
- @str : 文字列、
- @int : 整数、
- @bool : boolean 型、

- `@real` : 実数値、
- `@scalar` : 実数または複素数値、
- `@list` : リスト。

さらに、`@tobj` はインタフェースによって定義されているオブジェクトを参照します。たとえば、`@tmesh`, `@tmesh_fem`, `@tfem`, などを参照できます。いくつかの略語が使用可能です。

- `@tcont_struct` は `@tcs`
- `@tmesh_fem` は `@tmf`
- `@tgeotrans` は `@tgt`
- `@tglobal_function` は `@tgf`
- `@tmesher_object` は `@tmo`
- `@tmesh_levelset` は `@tmls`
- `@tmesh_im` は `@tmim`
- `@tlevelset` は `@tls`
- `@tslice` は `@tsl`
- `@tspmat` は `@tsp`
- `@tprecond` は `@tpre`

パラメータリスト (`...`) の末尾にある 3 つの点は、追加のパラメータが可能であることを意味します。省略可能なパラメーターは、角かっこで記述します。たとえば、 `/*@SET v = ('name'[, @int i])` です。しかし、python インターフェイスを構築する際には、 `extract_doc` スクリプトによってどのように解釈されるか注意してください。

マクロの 2 から 5 番目のパラメーターは、入力引数の最小数、最大値、出力引数の最小数、および出力引数の最大数にそれぞれ対応しています。これらは動的に検証されています。

関数リストの追加パラメータ...

原因不明の理由により、関数の本体には `int a, b;` などの複数の宣言を含めることができません (c++ でマクロの追加パラメーターであると解釈されてしまいます)。

- c++ の解説に含まれているドキュメントの部分は、 reStructuredText 形式である必要があります。特に、数式は :math:‘f(x) = 3x^2+2x+4‘ かもしくは次のように含めることができます。

```
.. math::

  f(x) = 3x^2+2x+4
```

記号 `INIT::OBJNAME('method-name', ...)`, `GET::OBJNAME('method-name', ...)`, `SET::OBJNAME('method-name', ...)`, `FUNC::FUNCNAME('subcommand-name', ...)` を使用して、インターフェイスの別のメソッドまたは関数を参照することができます。これは言語 (Matlab、Scilab または Python) に応じて正しく置き換えられます。

- ドキュメンテーションでは、特定の言語に応じて `@MATLAB{specific part ...}`, `@SCILAB{specific part ...}` および `@PYTHON{specific part ...}` を追加します。メソッド/サブコマンドがインタフェースに固有である場合、例えば Matlab の場合、 *MATLABFUNC* を使うことで *GET* を *MATLABGET*, *FUNC* などで置き換えることができます。この追加機能に特定のコードが必要な場合は、タグ `/*@MATLABEXT`, `/*@SCILABEXT`, `/*@PYTHONEXT` で追加することができます。たとえば、 `gf_mesh_fem_get.cc` を参照してください。
- Python と Matlab オブジェクトの場合、 *SET* メソッドの名前が *GET* メソッドと同じならば、 *SET* メソッドの先頭に *set_* が付けられます。

4.15.4 getfem インターフェイスへの新しい関数またはオブジェクトメソッドの追加

新しい関数 `gf_mesh_get(m, "foobar", .)`, を追加したい場合、 変更するメインファイルは `gf_mesh_get.cc` です。ユーザーがその関数を使用するときに scilab、matlab または python をクラッシュさせないようにするために、関数に渡されるすべての引数をチェックしてください。関数を追加するには、`gf_mesh_get.cc` で定義されているマクロを使用します。

`gf_mesh_get.cc` 内の関数のドキュメントを忘れずに追加してください。これは、 matlab/scilab/python のヘルプファイル (matlab のプロンプトで "`help gf_mesh_get`" とタイプしたときにヘルプファイルに表示されるドキュメントです)、 そして getfem_python の自動生成ドキュメントに反映されます。

重要。配列のインデックスは、Python では 0 から始まり、Matlab と Scilab では 1 から始まることに注意してください。次のような特定の関数の場合、

```
config::base_index()
```

インデックスとインデックスの配列を交換する際には python では 0、 Matlab と Scilab では 1 を使用しなければなりません。補正を 2 回しないように気をつけてください。インデックスのいくつかの配列は自動的にシフトされています。

4.15.5 getfem インターフェイスへの新しいオブジェクトの追加

インターフェイスに新しいオブジェクトを追加するには、それに対応する新しいソース `gf_obj.cc`, `gf_obj_get.cc` と `gf_obj_set.cc` を作成する必要があります。もちろん、モデルとして既存のものを使うこともできます。

オブジェクトを管理するために、 `getfemint.h` の最初にクラスを宣言する必要があります (アルファベット順になるように注意してください)。そして 3 つの関数を宣言します。

```
bool is_"name"_object(const mexarg_in &p);
id_type store_"name"_object(const std::shared_ptr<object_class> &shp);
object_class *to_"name"_object(const mexarg_in &p);
```

ここで、 "name" はインターフェイス内のオブジェクトの名前であり、 `object_class` は getfem のクラス名です (たとえば、メッシュオブジェクトの場合は `getfem::mesh`)。また、 *GetFEM++* で共有ポインタによって操作されるオブジェクトの場合、3 番目の関数は共有ポインタを返します。

重要: インタフェースを作成するためには、 *GetFEM++* オブジェクトは `dal::static_stored_object` を使用しなければなりません。一方、そのようにしない場合には、ラッパークラスを `bgeot::base_poly` のように定義します (`getfemint.h` の末尾を参照してください)。

前の 3 つの関数は、 `getfemint.cc` の末尾に実装する必要があります。 `getfemint.cc` で定義されている 2 つのマクロのいずれかを使用することができます。最初のマクロは標準的な目的のためであり、 *GetFEM++* の共有ポインタで操作するために 2 番目のオブジェクトを使用します。

`getfemint.cc` の最後に `name_of_getfemint_class_id` と `class_id_of_object` という関数も追加する必要があります。

`getfem_interface.cc` にインターフェイス関数の呼び出しを追加する必要があります。そしてファイル `bin/extract_doc` を変更し設定ファイルを実行します。

メソッド `get('char')` と `get('display')` は、各オブジェクトに対して定義する必要があります。最初のメソッドでは、文字列によりオブジェクトをファイルに保存するできるようにし、2 番目のメソッドでは文字列によりオブジェクトに関するいくつかの情報を与えるようにします。さらに、文字列による構築では、ファイルからオブジェクトを読み込む必要があります。

Scilab インターフェイスのためにファイル `sci_gateway/c/builder_gateway_c.sce.in` とディレクトリ `macros/overload` 内のファイルを変更する必要があります。

4.15.6 状態

4.15.7 展望

インターフェイスは *GetFEM++* と連動して成長しています。 *GetFEM++* の主な関数群のインタフェースは実装されています。

Chapter 5

付録A. 参照要素と実数要素の基本的な計算

5.1 体積積分

次式で与えられます。

$$\int_T f(x)\,dx = \int_{\widehat{T}} \widehat{f}(\widehat{x})|\text{vol}\left(\frac{\partial\tau(\widehat{x})}{\partial\widehat{x}_0};\frac{\partial\tau(\widehat{x})}{\partial\widehat{x}_1};\ldots;\frac{\partial\tau(\widehat{x})}{\partial\widehat{x}_{P-1}}\right)|\,d\widehat{x}.$$

$J_\tau(\widehat{x})$ は Jacobian を意味します

$$\boxed{J_\tau(\widehat{x}) := |\text{vol}\left(\frac{\partial\tau(\widehat{x})}{\partial\widehat{x}_0};\frac{\partial\tau(\widehat{x})}{\partial\widehat{x}_1};\ldots;\frac{\partial\tau(\widehat{x})}{\partial\widehat{x}_{P-1}}\right)| = (\det(K(\widehat{x})^T K(\widehat{x})))^{1/2},}$$

最終的に以下のように表されます。

$$\boxed{\int_T f(x)\,dx = \int_{\widehat{T}} \widehat{f}(\widehat{x}) J_\tau(\widehat{x})\,d\widehat{x}.}$$

$P = N$ のとき Jacobian の式は $J_\tau(\widehat{x}) = |\det(K(\widehat{x}))|$ のようになります。

5.2 面積分

Γ を実数要素である T の境界の一部、 $\widehat{\Gamma}$ を参照要素 $\widehat{T}$ 上の対応する境界とすると次式で表されます。

$$\boxed{\int_\Gamma f(x)\,d\sigma = \int_{\widehat{\Gamma}} \widehat{f}(\widehat{x})\|B(\widehat{x})\widehat{n}\|J_\tau(\widehat{x})\,d\widehat{\sigma},}$$

ここで、 $\widehat{n}$ は $\widehat{\Gamma}$ 上の $\widehat{T}$ の法線です。同様に次式が成り立ちます。

$$\boxed{\int_\Gamma F(x)\cdot n\,d\sigma = \int_{\widehat{\Gamma}} \widehat{F}(\widehat{x})\cdot(B(\widehat{x})\cdot\widehat{n})J_\tau(\widehat{x})\,d\widehat{\sigma},}$$

n は Γ 上の T に対する単位法線です。

5.3 微分計算

次式で与えられます。

$$\nabla f(x) = B(\widehat{x})\widehat{\nabla}\widehat{f}(\widehat{x}).$$

5.4 2階微分計算

使用するのは次の

$$\nabla^2 f = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{ij},$$

$N \times N$ 行列と

$$\widehat{X}(\widehat{x}) = \sum_{k=0}^{N-1} \widehat{\nabla}^2 \tau_k(\widehat{x}) \frac{\partial f}{\partial x_k}(x) = \sum_{k=0}^{N-1} \sum_{i=0}^{P-1} \widehat{\nabla}^2 \tau_k(\widehat{x}) B_{ki} \frac{\partial \widehat{f}}{\partial \widehat{x}_i}(\widehat{x}),$$

$P \times P$ 行列、そして

$$\widehat{\nabla}^2 \widehat{f}(\widehat{x}) = \widehat{X}(\widehat{x}) + K(\widehat{x})^T \nabla^2 f(x) K(\widehat{x}),$$

です。したがって展開すると次式になります。

$$\nabla^2 f(x) = B(\widehat{x})(\widehat{\nabla}^2 \widehat{f}(\widehat{x}) - \widehat{X}(\widehat{x}))B(\widehat{x})^T.$$

基本行列の計算を一様な方法で計算するために、Hessian の成分は $\frac{\partial^2 f}{\partial x_0^2}, \frac{\partial^2 f}{\partial x_1 \partial x_0}, \ldots, \frac{\partial^2 f}{\partial x_{N-1}^2}$ である列ベクトル Hf として計算されます。また、 B_2 は $P^2 \times P$ 行列で次のように定義されます。

$$[B_2(\widehat{x})]_{ij} = \sum_{k=0}^{N-1} \frac{\partial^2 \tau_k(\widehat{x})}{\partial \widehat{x}_{i/P} \partial \widehat{x}_{i \bmod P}} B_{kj}(\widehat{x}),$$

B_3 は $N^2 \times P^2$ 行列で次式のように定義されています

$$[B_3(\widehat{x})]_{ij} = B_{i/N, j/P}(\widehat{x}) B_{i \bmod N, j \bmod P}(\widehat{x}),$$

次式で表されます。

$$\boxed{Hf(x) = B_3(\widehat{x}) \left(\widehat{H}\ \widehat{f}(\widehat{x}) - B_2(\widehat{x}) \widehat{\nabla} \widehat{f}(\widehat{x})\right).}$$

5.5 基本的な行列の例

基本的な “行列” を計算する場合を想定します。

$$t(i_0, i_1, \ldots, i_7) = \int_T \varphi_{i_1}^{i_0} \partial_{i_4} \varphi_{i_3}^{i_2} \partial^2_{i_7/P, i_7 \bmod P} \varphi_{i_6}^{i_5}\, dx,$$

参照要素に対して行われる計算は次式

$$\widehat{t_0}(i_0, i_1, \ldots, i_7) = \int_{\widehat{T}} (\widehat{\varphi})_{i_1}^{i_0} \partial_{i_4} (\widehat{\varphi})_{i_3}^{i_2} \partial^2_{i_7/P, i_7 \bmod P} (\widehat{\varphi})_{i_6}^{i_5} J(\widehat{x})\, d\widehat{x},$$

と次式です。

$$\widehat{t_1}(i_0, i_1, \ldots, i_7) = \int_{\widehat{T}} (\widehat{\varphi})_{i_1}^{i_0} \partial_{i_4} (\widehat{\varphi})_{i_3}^{i_2} \partial_{i_7} (\widehat{\varphi})_{i_6}^{i_5} J(\widehat{x})\, d\widehat{x},$$

これらの 2 つのテンソルは、幾何変換が線形の場合、($J(\widehat{x})$ が一定であるため)、参照要素全体で一度計算できます。幾何変換が非線形である場合、格納されるべきものは各積分点の値です。実数要素上の積分を計算するには、いくつかの低減が必要です。

- 最初の項 ($\varphi_{i_1}^{i_0}$) については何もしません。
- 第 2 項 ($\partial_{i_4}\varphi_{i_3}^{i_2}$) は行列 B を使って i_4 とします。
- 第 3 項（ $\partial^2_{i_7/P,i_7} \bmod {}_P\varphi_{i_6}^{i_5}$ ）について $\widehat{t_0}$ の縮約行列 B_3 をもつ i_7 と行列 B_3B_2 をもつ i_7 に関する $\widehat{t_1}$ の縮約を考えます

幾何変換が非線形である場合には、各積分点に対して低減を行います。これらの低減が行われると、それらの縮約の結果として生じるすべてのテンソルが（幾何変換が非線形である場合、各積分点の負荷に等しい因子で）追加されます。

有限要素が τ- 等価でない場合、結果のテンソルを行列 M で補助的に低減しなければなりません。

Chapter 6

References

Bibliography

[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 implicitely 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. Poulios, 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.

[Nackenhorst2004] U. Nackenhorst, *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. Doležel, *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