Python Interface

リリース ***5.3***

Luis Saavedra

2019 年 **09** 月 **09** 日

目次

第 1 章

はじめに

このガイドでは、 *GetFEM++* の *Python* インターフェイスについて説明します。 *GetFEM++* を全て参照するには、 特定のガイド を参照してください。しかし、 *getfem-interface* の内部構造についての特別な知識がなくても、 *GetFEM++* の内部構造についての基本的な知識は必要ですが、 *getfem-interface* を使用できるはずです。ただし、このドキュメントは独立したものではありません。使用される構造体のアルゴリズムと概念についてのより詳細な説明は、主に user documentation を参照してください。

第 2 章

インストール

ソースからインストールする場合は *configure* スクリプトの *–enable-python* オプションを使用します。

インターフェイスの並列版については ud-parallel も参照してください。

GetFEM++ のインストールについては ダウンロードとインストール を参照してください。

第 3 章

準備

これは、このマニュアルで使用されている用語の簡単な要約です。有限要素に慣れていない場合であれば便利です (但し、どんな場合でも dp は読みましょう)。

mesh は convexes から構成されます。凸包と呼ばれるものは、単純な線分、角柱、四面体、曲三角形であり、(幾何学的な意味で) 凸包でないものであってもかまいません。これらはすべて関連する reference convex: セグメントの場合は $[0,1]$ セグメント、三角形の場合は正準三角形 $(0,0)-(0,1)-(1,0)$ などになります。メッシュのすべての凸包は、参照凸包から geometric transformation を介して構築されます。単純な場合 (例えば凸包が単純な場合)、この変換は (簡単に逆変換できることが大きな利点となる) 線形になります。幾何学的変換を定義するために、基準凸包上に geometrical nodes を定義します。幾何変換はこれらのノードを mesh nodes にマップします。

メッシュ上で、基底関数の集合 FEM を定義します。FEM を各凸包に関連付けます。基底関数は、一部の (自由に選ぶことができる) 形状点にもアタッチされます。これらのポイントはメッシュノードに似ていますが、 **同じである必要はありません** (これは、古典的な P_1 三角形メッシュ有限要素法のような非常に単純な場合にのみ発生します)。メッシュ上のすべての基底関数の集合は、偏微分方程式が解かれるベクトル空間の基底を形成します。これらの基底関数 (および関連する形状点) は **自由度** (dof に関連) します。FEM は各基底関数が積分された形状点で 1 に等しく、他の基底関数の形状点で null であるとき、 Lagrangian であるといいます。有限要素空間上の任意の関数を **補間** することが非常に簡単になるため、これは重要な特性です。

有限要素法は凸包 (の面) 上のこれらの基底関数 (基本関数などの積) の積分の評価を含みます。(多項式基底関数や線形幾何変換などの) 単純な場合では、これらの積分を解析的に評価することができます。他の場合には、以下を用いて近似しなければなりません。したがって、各凸包には FEM とともに **積分法** の **求積公式** の式が付加されます 。近似積分法を使用する必要がある場合は、常にその次数 (すなわち、方法と正確に積分された多項式の最高次数) を慎重に選択してください。FEM の次数、幾何変換の多項式次数、および基本行列の性質を考慮する必要があります。適切な次数がわからない場合は、不要な線形システムを生成する低次積分法よりも、(アセンブリが遅くなりますが) 常に高次積分法を選択します。

各凸包の基底関数の積分から全体線形システムを構築するプロセスは `assembly` です。

一連の FEM が凸包に接続されたメッシュは *GetFEM++* で `mesh_fem` オブジェクトと呼ばれます。

一連の積分法が接続されたメッシュは *GetFEM++* で `mesh_im` オブジェクトと呼ばれます。

mesh_fem を使用してスカラーフィールド (熱、圧力、...) またはベクトルフィールド (変位、電場、...) を近似できます。 *mesh_im* はこれらのフィールドで数値積分を実行するために使用されます。 *GetFEM++* に実装されているほとんどの有限要素はスカラー (ただし TR_0 と edges 要素も利用可能) です。もちろん、これらのスカラー FEM を使用して、ベクトル場の各成分を近似することができます。これを行うには、*mesh_fem* の $Qdim$ をベクトルフィールドの次元に設定します (例えば $Qdim = 1 \Rightarrow$ スカラーフィールド、 $Qdim = 2 \Rightarrow$ ベクトルフィールドなど)。

偏微分方程式を解く場合、多くの場合、複数の FEM を使用する必要があります。もちろん偏微分方程式の解が定義されている問題は最も重要です。ただし、ほとんどの偏微分方程式にはさまざまな係数が含まれます。たとえば、次のような場合です。

$$\nabla \cdot (\lambda(x)\nabla u) = f(x).$$

したがって、メインの未知数の u に対して FEM を定義する必要もありますが、定数でない場合 $\lambda(x)$ と $f(x)$ に対しても定義が必要になります。これらの係数を有限要素空間で容易に補間するために Lagrange 有限要素法を選択することが多いです。

凸包、メッシュ節点、および自由度にはすべて番号が付けられます。凸包に関連する数をその数 **凸包 id** (`cvid` とします) で参照することがあります。メッシュ節点番号は **ポイント id** (`pid` とします) とも呼ばれます。凸包面にはグローバルな番号付けはなく、各凸包のローカルな番号のみが付けられます。したがって、面のリストを必要とする関数や返す関数は、常に 2 行の行列を使用します。最初の行列には凸包 id が含まれ、2 番目の行列にはローカル面番号が含まれます。

自由度は常に連続した番号が付けられますが、 **点の ID と凸包の ID に対しては常に成り立つわけではありません** 。特に、メッシュから点や凸包を削除した場合は成り立ちません。これらが連続した配列 (1 から開始) を形成することを保証するためには、次のように実行する必要があります。

```
>>> m.set('optimize structure')
```

第 4 章

Python GetFEM++ インターフェイス

4.1 はじめに

GetFEM++ は、スクリプト言語 *Python* へのインターフェイスを提供します。 *Python* はクロスプラットフォームでフリーな言語です。numpy パッケージの追加により、python は Matlab 機能のサブセット (すなわち密配列) を提供します。VTK ツールキットは python インタフェース (または MayaVi) を介して視覚化ツールを提供し、 OpenDX のデータファイルをエクスポートすることができます。ただし、このガイドでは、結果を視覚化するために、Gmsh ポスト処理フォーマットにエクスポートします。疎行列ルーチンは getfem インタフェースによって提供されます。

python インタフェースは python モジュール getfem.py を介して使用できます。インターフェースを使用するには、次のようにロードする必要があります。

```
import getfem
m = getfem.Mesh('cartesian', range(0, 3), range(0,3))
```

または

```
from getfem import *
m = Mesh('cartesian', range(0, 3), range(0,3))
```

getfem.py (と内部の _getfem.so) モジュールが標準の場所にインストールされていないと、環境変数 PYTHONPATH をその場所に設定する必要があります。次に例を示します。

```
import sys
sys.path.append('.../getfem/getfem++/interface/src/python/')
```

4.2 並列版

現時点では、mpi ベースの Getfem 並列版をインターフェースしているのは Python インターフェースだけです。 ud-parallel を参照してください。

4.3 メモリ管理

Matlab インターフェースの優れた利点は、使用されなくなったオブジェクトを明示的に削除する必要がなく、自動的に削除されることです。しかし、getfem ワークスペースの内容は、 `getfem.memstats()` 関数で調べることができます。

4.4 文書化

getfem モジュールについては多くの文書があります。このドキュメントは *API リファレンス* に展開されています。getfem-matlab のユーザーガイドも使用できますが、これは、このガイドの内容の 95% が Python に直接変換されているためです (ただし、matlab に固有の描画関数は例外です)。

4.5 *Python GetFEM++* 構成

python-interface の一般的な構成は次のとおりです。

- matlab インターフェースの各クラスは、Python インターフェースに対応するクラスを持っています。Python の Mesh クラスでは、 gfSlice が getfem.Slice などになります。
- matlab インターフェースの各 get および set メソッドは、Python インターフェースの対応するクラスのメソッドに変換されています。次に例を示します。

```
gf_mesh_get(m, 'outer faces');
gf_mesh_get(m, 'pts');
```

次のようになります。

```
m.outer_faces();
m.pts();
```

曖昧さがあったときに名前が変更されたメソッドもあります。例えば `gf_mesh_set(m, 'pts', P)` は `m.set_pts(P)` です。

- もう 1 つの getfem-matlab 関数には、対応する Python 関数への非常に単純なマッピングがあります。

gf_compute(mf,U,'foo',...)	getfem.compute_foo(mf,U) または getfem.compute('foo',...)
gf_asm('foobar',...)	getfem.asm_foobar(...) または getfem.asm('foobar',...)
gf_linsolve('gmres',...)	getfem.linsolve_gmres(...) または getfem.linsolve('gmres',...)

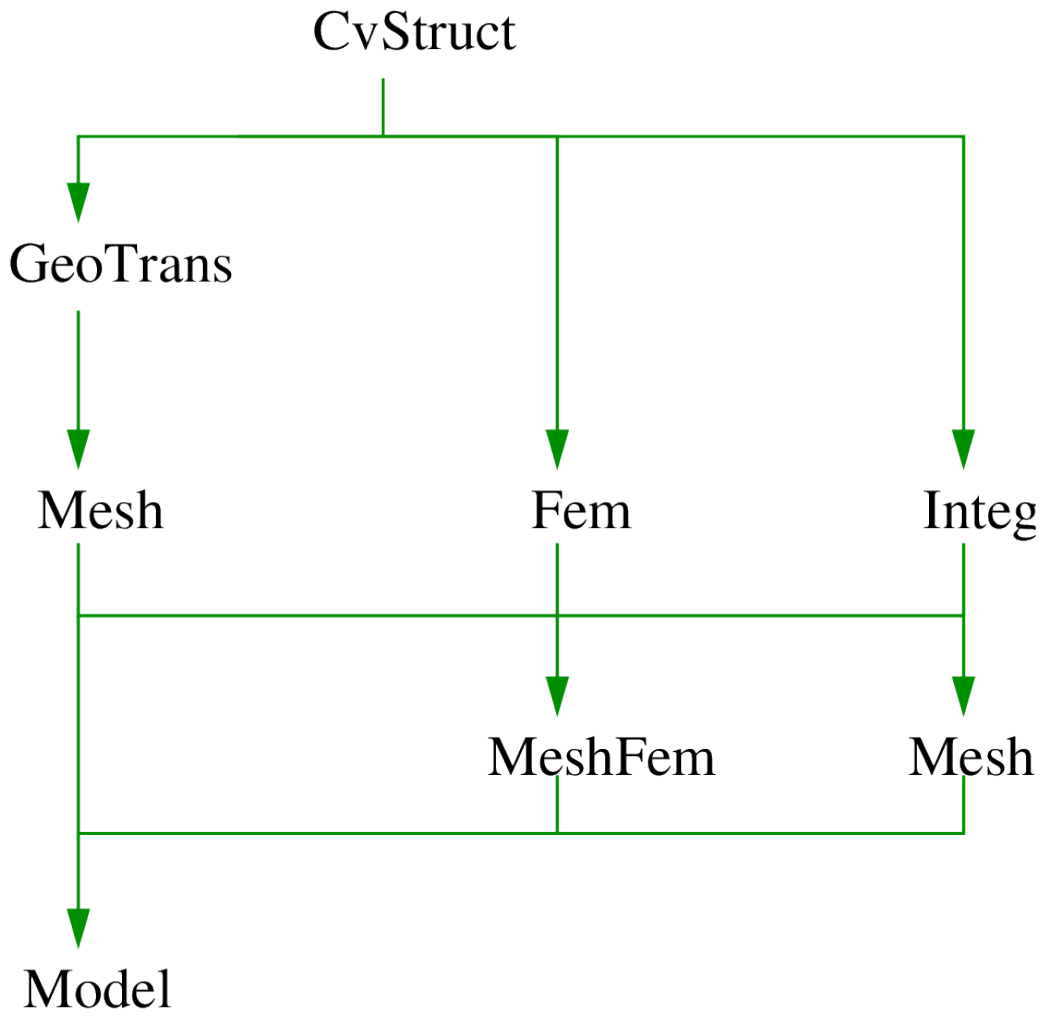

図 1　python-getfem インタフェースメインオブジェクトの階層。

class CvStruct (*self*, **args*)

凸包構造オブジェクトの記述子は凸包構造の形式情報 (凸包の構造である点の数、面の数) を格納します。

class GeoTrans (*self*, **args*)

幾何変換オブジェクト (凸包の形状/位置を定義する) 記述子。

class Mesh (*self*, **args*)

メッシュ構造 (節点、凸包、各凸包の幾何変換) 記述子。

class Fem(*self*, *fem_name*)

FEM(有限要素法) オブジェクト の記述子 (1 つの凸包ごとに PK, QK, HERMITE, などがあります)。

class Integ(*self*, **args*)

積分法オブジェクト の記述子 (正確な求積公式 ldots)。GeoTrans に直接リンクされているわけではありませんが、通常、積分法は特定の凸包構造に固有です。

class MeshFem(*self*, **args*)

メッシュにリンクされたオブジェクト の記述子です。メッシュの各凸包には FEM が割り 当てられています。

class MeshIm(*self*, **args*)

メッシュにリンクされたオブジェクト の記述子です。各凸包には積分法が割り 当てられています。

class Model(*self*, **args*)

model オブジェクト の記述子で、モデルのグローバルデータ、変数、および記述子を 保持します。 *GetFEM++* のバージョン 4.0 では *model state* および *model brick* オブジェクト が進化しました。

第 5 章

例題

5.1 ステップごとの基本的な例

この例は、以下のような über-canonical 問題に対する getfem の基本的な使い方を示しています。これは Laplacian, $-\Delta u = f$ を正方形上で、Dirichlet 条件 $u = g(x)$ をドメイン境界上で解きます。この例の **py-file** は **demo_step_by_step.py** という名前の下にあります。 *GetFEM++* ディストリビューションの `interface/tests/python/` ディレクトリにあります。

最初のステップは **メッシュオブジェクトを作成する** ことです。単純なジオメトリ (`getfem.Mesh('generate', mesher_object mo, scalar h))` を参照) 用に単純な構造化メッシュまたは非構造化メッシュを作成することも、外部メッシャ (`getfem.Mesh('import', string FORMAT, string FILENAME)` を参照) に依存することも、非常に単純なメッシュを使用することも可能です。この例では、ノードが $\{x_{i=0...10,j=0..10} = (i/10, j/10)\}$ である通常の mesh index{cartesian mesh} を考えます。

```
# import basic modules
import getfem as gf
import numpy as np

# creation of a simple cartesian mesh
m = gf.Mesh('cartesian', np.arange(0,1.1,0.1), np.arange(0,1.1,0.1))
```

次のステップでは **MeshFem オブジェクト** を作成します。これは、メッシュを FEM の集合とリンクします。

```
# create a MeshFem of for a field of dimension 1 (i.e. a scalar field)
mf = gf.MeshFem(m, 1)
# assign the Q2 fem to all convexes of the MeshFem
mf.set_fem(gf.Fem('FEM_QK(2,2)'))
```

最初の命令は新しい MeshFem オブジェクトを構築し、2 番目の引数はこのオブジェクトがスカラー

フィールドの補間に使用されることを指定します (未知変数 u はスカラーフィールドです)。第 2 の命令は、すべての凸包 (各基底関数は 4 次の多項式であり k 次 $P^k \Rightarrow$ 多項式であることを覚えておいてください、ここで $2k$ 次 $Q^k \Rightarrow$ 多項式です) に Q^2 FEM を割り当てます。 Q^2 は多項式 FEM なので、参照凸包でその基底関数の式を見ることができます。

```
# view the expression of its basis functions on the reference convex
print gf.Fem('FEM_QK(2,2)').poly_str()
```

さて、 mf で数値積分を行うには、 **MeshIm オブジェクトを構築する** 必要があります。

```
# an exact integration will be used
mim = gf.MeshIm(m, gf.Integ('IM_EXACT_PARALLELEPIPED(2)'))
```

積分法を使用して各要素の各種積分を計算します。ここでは、正確な計算 (非 **求積公式**) を実行することを選択します。これは、参照凸包からのこれらの凸包の幾何変換が線形 (これはすべてのシンプレックスに当てはまり、通常のメッシュの平行六面体にも当てはまりますが、一般的な四角形には当てはまりません) であり、選択した FEM が多項式であるために可能です。このため、すべての基底関数/基底関数の積/傾き/などを解析的に積分することが可能であり、FEM の手法や積分法にも多くの選択肢があります (ud を参照)。

ただし、一般的なケースでは、厳密な積分法よりも近似積分法の方が適しています。

次に、Dirichlet 条件を設定するために、**領域の** <boundary> を **検索する** 必要があります。メッシュオブジェクトには、凸包面と凸包面の集合を保存する機能があります。これらの集合 (<regions> と呼ばれます) には、整数 *#id* を使用してアクセスします。

```
# detect the border of the mesh
border = m.outer_faces()
# mark it as boundary #42
m.set_region(42, border)
```

ここでは、メッシュの境界上にある凸包の面 (つまり、2 つの凸包によって共有されていない面) を見つけます。

配列 border には二つの行があり、最初の行は凸包の番号で、次の行は面の番号です (これは凸包に対してローカルであり、面のグローバルな番号付けはありません)。次に、この面の集合を領域番号 42 に割り当てます。

この時点では、モデルを記述し、ソルバを実行して解を取得するだけです。 "model" は Model コンストラクタで作成されます。モデルは基本的に、全体線形システム (非線形問題の接線行列) とそれに関連する RHS を構築するオブジェクトです。代表的な修正は、考慮される問題 (線形弾性、Laplacian 等) に対する剛性マトリックスの挿入、拘束の集合の処理、Dirichlet 条件、ソース項の RHS への追加などです。全体正接行列とその右辺は、 "model" 構造に格納されます。

簡単な方法で問題を作成してみましょう。 $u = x(x-1) - y(y-1)$ とすると、 $-\Delta u = 0$ (Q^2 法を使っているので、FEM は正確な解を捕らえることができません) となります。

まず空の実数モデルから始めます。

```
# empty real model
md = gf.Model('real')
```

(モデルは `'real'` か `'complex'` のどちらかです)。そして有限要素法 *mf* のシステムでは、 `u` は未知であると宣言します。

```
# declare that "u" is an unknown of the system
# on the finite element method `mf`
md.add_fem_variable('u', mf)
```

ここで、以下の問題を処理する *generic elliptic* ブリックを追加します。ここで $-\nabla \cdot (A : \nabla u) = \ldots$ 問題をハンドルします。ここで A はスカラー場、行列場、4 次テンソル場の場合があります。デフォルトでは、 $A = 1$ です。これを変数 `u` に追加します。

```
# add generic elliptic brick on "u"
md.add_Laplacian_brick(mim, 'u');
```

次に、領域の境界に Dirichlet 条件を追加します。

```
# add Dirichlet condition
g = mf.eval('x*(x-1) - y*(y-1)')
md.add_initialized_fem_data('DirichletData', mf, g)
md.add_Dirichlet_condition_with_multipliers(mim, 'u', mf, 42, 'DirichletData
↪')
```

最初の 2 行は、Dirichlet 条件の値を表すモデルのデータを定義します。3 つ目は、境界番号 `42` の変数 `u` に Dirichlet 条件を追加します。Dirichlet 条件は Lagrange 乗数を用いて課しました。他にはペナルティ法があります。 MeshFem 引数も必要です。これは Dirichlet 条件 $u = g$ が弱形式 $\int_\Gamma u(x)v(x) = \int_\Gamma g(x)v(x)\ \forall v$ で課されるからです。ここで v はここでは `mf` で与えられる乗数空間で使われます。

備考:

多項式は `mf` で補間されます。これは `mf` が Lagrange 型の場合のみ可能です。この最初の例では、未知変数と `g` のようなデータに同じ MeshFem を使用していますが、一般的なケースでは `mf` は Lagrangian ではなく、Dirichlet 条件やソース項などの記述には別の (Lagrangian) MeshFem が使用されます。

ソース項は以下の行 (コメント なし) で追加できます。

```
# add source term
#f = mf.eval('0')
#md.add_initialized_fem_data('VolumicData', mf, f)
#md.add_source_term_brick(mim, 'u', 'VolumicData')
```

残るはソルバを起動するだけです。次の命令で線形系をアセンブルして解きます。

```
# solve the linear system
md.solve()
```

これで、(解かれた線形系のような他のものと同様に) モデルに解が含まれました。次のように抽出します。

```
# extracted solution
u = md.variable('u')
```

次に、解をエクスポートします

```
# export computed solution
mf.export_to_pos('u.pos',u,'Computed solution')
```

そして `gmsh u.pos` で見てください。図 計算された解 を参照。

5.2 別の Laplacian による厳密解 (ソース項)

この例では、正規問題での getfem の基本的な使い方を示しています。正規問題とは、四角形上で Laplacian、 $-\Delta u = f$ を解くことです、ここで、Dirichlet 条件は Γ_D ドメイン境界上で $u = g(x)$ 、Neumann 条件は Γ_N ドメイン境界上で $\frac{\partial u}{\partial \eta} = h(x)$ です。この例の **py-file** は *GetFEM++* ディストリビューションの `interface/tests/python/` ディレクトリに **demo_laplacian.py** という名前であります。

Mesh、MeshFem、MeshIm オブジェクトを作成し、前の例と同じ方法で領域の境界を見つけます。

```
# import basic modules
import getfem as gf
import numpy as np

# boundary names
top   = 101 # Dirichlet boundary
down  = 102 # Neumann boundary
left  = 103 # Dirichlet boundary
```

(次のページに続く)

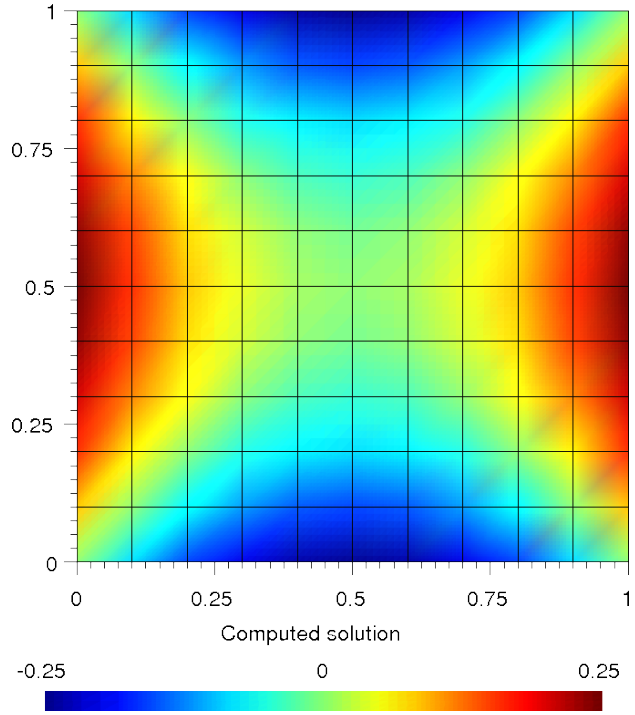

図 1　計算された解

(前のページからの続き)

```
right = 104 # Neumann boundary

# parameters
NX = 40                                # Mesh parameter
Dirichlet_with_multipliers = True;  # Dirichlet condition with multipliers␣
↪or penalization
dirichlet_coefficient = 1e10;       # Penalization coefficient

# mesh creation
m = gf.Mesh('regular_simplices', np.arange(0,1+1./NX,1./NX), np.arange(0,1+1.
↪/NX,1./NX))

# create a MeshFem for u and rhs fields of dimension 1 (i.e. a scalar field)
mfu   = gf.MeshFem(m, 1)
mfrhs = gf.MeshFem(m, 1)
# assign the P2 fem to all convexes of the both MeshFem
mfu.set_fem(gf.Fem('FEM_PK(2,2)'))
mfrhs.set_fem(gf.Fem('FEM_PK(2,2)'))

# an exact integration will be used
mim = gf.MeshIm(m, gf.Integ('IM_TRIANGLE(4)'))
```

(次のページに続く)

(前のページからの続き)

```

# boundary selection
flst    = m.outer_faces()
fnor    = m.normal_of_faces(flst)
ttop    = abs(fnor[1,:]-1) < 1e-14
tdown   = abs(fnor[1,:]+1) < 1e-14
tleft   = abs(fnor[0,:]+1) < 1e-14
tright  = abs(fnor[0,:]-1) < 1e-14
ftop    = np.compress(ttop, flst, axis=1)
fdown   = np.compress(tdown, flst, axis=1)
fleft   = np.compress(tleft, flst, axis=1)
fright  = np.compress(tright, flst, axis=1)

# mark it as boundary
m.set_region(top, ftop)
m.set_region(down, fdown)
m.set_region(left, fleft)
m.set_region(right, fright)
```

次に、厳密解とソース項を補間します。

```
# interpolate the exact solution (assuming mfu is a Lagrange fem)
g = mfu.eval('y*(y-1)*x*(x-1)+x*x*x*x*x')

# interpolate the source terms (assuming mfrhs is a Lagrange fem)
f = mfrhs.eval('-(2*(x*x+y*y)-2*x-2*y+20*x*x*x)')
h = mfrhs.eval('[y*(y-1)*(2*x-1) + 5*x*x*x*x, x*(x-1)*(2*y-1)]')
```

前の例のように問題をブリックにしました

```
# model
md = gf.Model('real')

# add variable and data to model
md.add_fem_variable('u', mfu)               # main unknown
md.add_initialized_fem_data('f', mfrhs, f) # volumic source term
md.add_initialized_fem_data('g', mfrhs, g) # Dirichlet condition
md.add_initialized_fem_data('h', mfrhs, h) # Neumann condition

# bricked the problem
md.add_Laplacian_brick(mim, 'u')                                # laplacian␣
↪term on u
md.add_source_term_brick(mim, 'u', 'f')                         # volumic␣
↪source term
md.add_normal_source_term_brick(mim, 'u', 'h', down)            # Neumann␣
↪condition
```

(次のページに続く)

(前のページからの続き)

```
md.add_normal_source_term_brick(mim, 'u', 'h', left)          # Neumann
↪condition

# Dirichlet condition on the top
if (Dirichlet_with_multipliers):
  md.add_Dirichlet_condition_with_multipliers(mim, 'u', mfu, top, 'g')
else:
  md.add_Dirichlet_condition_with_penalization(mim, 'u', dirichlet_
↪coefficient, top, 'g')

# Dirichlet condition on the right
if (Dirichlet_with_multipliers):
  md.add_Dirichlet_condition_with_multipliers(mim, 'u', mfu, right, 'g')
else:
  md.add_Dirichlet_condition_with_penalization(mim, 'u', dirichlet_
↪coefficient, right, 'g')
```

唯一の変更点は **ソース項** ブリックの追加です。最後に問題の解を抽出しエクスポートします。

```
# assembly of the linear system and solve.
md.solve()

# main unknown
u  = md.variable('u')
L2error = gf.compute(mfu, u-g, 'L2 norm', mim)
H1error = gf.compute(mfu, u-g, 'H1 norm', mim)

if (H1error > 1e-3):
    print 'Error in L2 norm : ', L2error
    print 'Error in H1 norm : ', H1error
    print 'Error too large !'

# export data
mfu.export_to_pos('sol.pos', g,'Exact solution',
                             u,'Computed solution')
```

`gmsh sol.pos` で見てください。

5.3 線形および非線形弾性

この例では、 GiD で生成されたメッシュを使用しています。オブジェクトは 2 次 4 面体でメッシュ分割されます。この例の **py-file** は *GetFEM++* ディストリビューションの `interface/tests/python/` ディレクトリの `demo_tripod.py` にあります。

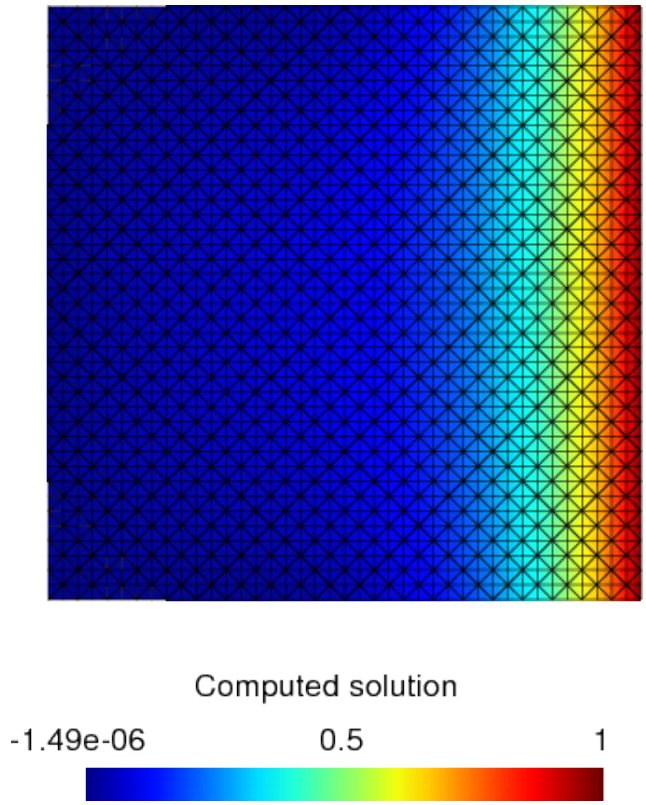

図 2　差分

```
import getfem as gf
import numpy as np

# parameters
file_msh = 'tripod.GiD.msh'
degree = 2
linear = False
incompressible = False # ensure that degree > 1 when incompressible is on..
E = 1e3
Nu = 0.3
Lambda = E*Nu/((1+Nu)*(1-2*Nu))
Mu = E/(2*(1+Nu))

# create a Mesh object (importing)
m = gf.Mesh('import','gid',file_msh)
m.set('optimize_structure')

# create a MeshFem object
mfu = gf.MeshFem(m,3) # displacement
```

(次のページに続く)

(前のページからの続き)

```
mfp = gf.MeshFem(m,1) # pressure
mfe = gf.MeshFem(m,3) # for plot displacement
mff = gf.MeshFem(m,1) # for plot von-mises
# assign the FEM
mfu.set_fem(gf.Fem('FEM_PK(3,%d)' % (degree,)))
mfp.set_fem(gf.Fem('FEM_PK_DISCONTINUOUS(3,0)'))
mfe.set_fem(gf.Fem('FEM_PK_DISCONTINUOUS(3,1,0.01)'))
mff.set_fem(gf.Fem('FEM_PK_DISCONTINUOUS(3,1,0.01)'))

# build a MeshIm object
mim = gf.MeshIm(m,gf.Integ('IM_TETRAHEDRON(5)'))

print 'nbcvs=%d, nbpts=%d, qdim=%d, fem = %s, nbdof=%d' % \
      (m.nbcvs(),m.nbpts(),mfu.qdim(),mfu.fem()[0].char(),mfu.nbdof())

# detect some boundary of the mesh
P = m.pts()
ctop = (abs(P[1,:] - 13) < 1e-6)
cbot = (abs(P[1,:] + 10) < 1e-6)
pidtop = np.compress(ctop,range(0,m.nbpts()))
pidbot = np.compress(cbot,range(0,m.nbpts()))
ftop = m.faces_from_pid(pidtop)
fbot = m.faces_from_pid(pidbot)
# create boundary region
NEUMANN_BOUNDARY = 1
DIRICHLET_BOUNDARY = 2
m.set_region(NEUMANN_BOUNDARY,ftop)
m.set_region(DIRICHLET_BOUNDARY,fbot)

# the model bricks
if linear:
  b0 = gf.MdBrick('isotropic_linearized_elasticity',mim,mfu)
  b0.set_param('lambda',Lambda)
  b0.set_param('mu',Mu)
  if (incompressible):
    b1 = gf.MdBrick('linear_incompressibility term',b0,mfp)
  else:
    b1 = b0
else:
  # large deformation with a linearized material law.. not a very good␣
↪choice!
  if (incompressible):
    b0 = gf.MdBrick('nonlinear_elasticity',mim,mfu,'Mooney_Rivlin')
    b0.set_param('params',[Lambda,Mu])
    b1 = gf.MdBrick('nonlinear_elasticity_incompressibility_term',b0,mfp)
  else:
```

(次のページに続く)

(前のページからの続き)

```
    b0 = gf.MdBrick('nonlinear_elasticity',mim,mfu,'SaintVenant_Kirchhoff')
    #b0 = gf.MdBrick('nonlinear_elasticity',mim,mfu,'Ciarlet_Geymonat')
    b0.set_param('params',[Lambda,Mu])
    b1 = b0

b2 = gf.MdBrick('source_term',b1,NEUMANN_BOUNDARY)
b2.set_param('source_term',[0,-10,0])
b3 = gf.MdBrick('dirichlet',b2,DIRICHLET_BOUNDARY,mfu,'penalized')

# create model state
mds = gf.MdState(b3)
# running solve...
b3.solve(mds,'noisy','lsolver','superlu')

# extracted solution
U = mds.state()

# post-processing
VM = b0.von_mises(mds,mff)

# export U and VM in a pos file
sl = gf.Slice(('boundary',),mfu,1)
sl.export_to_pos('sol.pos', mfu, U, 'Displacement', mff, VM, 'Von Mises␣
↪Stress')
```

次の図は Von Mises 応力および変位ノルムを示しています。

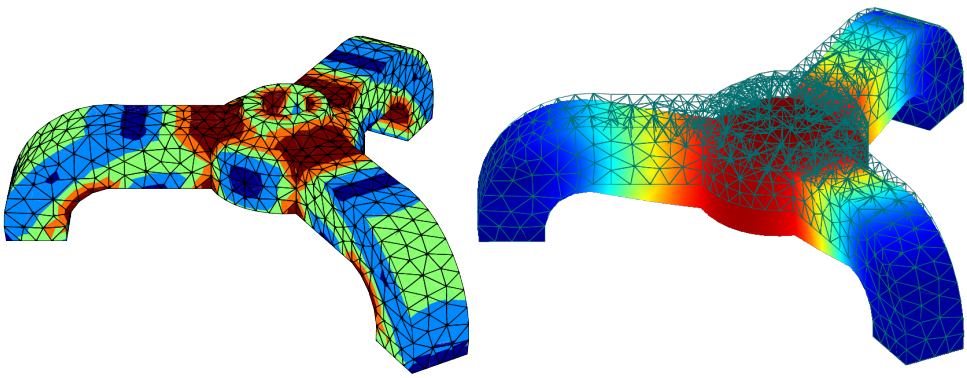

図 3　(a) 三脚 Von Mises, (b) 三脚変位ノルム。

5.4 モデルフレームワークを使わない

モデルブリックは、最終的な線形システムのアセンブリのほとんどの詳細を隠蔽するため、非常に便利です。しかし、より低い水準で、線形システムのアセンブリと解析を直接 *Python* で処理することも可能です。例えば、 `demo_tripod_alt.py` はアセンブリが明示的であることを除き、`demo_tripod.py` によく似ています。

```
# create a MeshFem object
mfd = gf.MeshFem(m,1) # data
# assign the FEM
mfd.set_fem(gf.Fem('FEM_PK(3,0)'))

# assembly
nbd = mfd.nbdof()
F = gf.asm_boundary_source(NEUMANN_BOUNDARY, mim, mfu, mfd, np.repeat([[0],[-
↪100],[0]],nbd,1))
K = gf.asm_linear_elasticity(mim, mfu, mfd, np.repeat([Lambda], nbd), np.
↪repeat([Mu], nbd))

# handle Dirichlet condition
(H,R) = gf.asm_dirichlet(DIRICHLET_BOUNDARY, mim, mfu, mfd, mfd.eval('numpy.
↪identity(3)'), mfd.eval('[0,0,0]'))
(N,U0) = H.dirichlet_nullspace(R)

Nt = gf.Spmat('copy',N)
Nt.transpose()
KK = Nt*K*N
FF = Nt*F # FF = Nt*(F-K*U0)

# solve ...
P = gf.Precond('ildlt',KK)
UU = gf.linsolve_cg(KK,FF,P)
U = N*UU+U0

# post-processing
sl = gf.Slice(('boundary',), mfu, degree)

# compute the Von Mises Stress
DU = gf.compute_gradient(mfu,U,mfe)
VM = np.zeros((DU.shape[2],),'d')
Sigma = DU

for i in range(DU.shape[2]):
  d = np.array(DU[:,:,i])
  E = (d+d.T)*0.5
  Sigma[:,:,i]=E
```

(次のページに続く)

(前のページからの続き)

```
  VM[i] = np.sum(E**2) - (1./3.)*np.sum(np.diagonal(E))**2

SigmaSL = gf.compute_interpolate_on(mfe,Sigma,sl)

# export to Gmsh
sl.export_to_pos('tripod.pos', mfe, VM,'Von Mises Stress', mfu, U,
↪'Displacement')
sl.export_to_pos('tripod_ev.pos', mfu, U, 'Displacement', SigmaSL, 'stress')
```

getfem-interface では、ベクトルと行列のアセンブリは `gf.asm_*` 関数により行われます。Dirichlet 条件 $h(x)u(x) = r(x)$ は、弱形式 $\int (h(x)u(x)).v(x) = \int r(x).v(x) \quad \forall v$ で処理されます (ここで、$h(x)$ は 3×3 行列です。これは定数であり、単位と等価です)。元のシステムから Dirichlet 条件を除去することにより、縮退システム `KK UU = FF` を構築します。ペナルティ法を使用して Dirichlet 条件を処理する方が効率的 (よりシンプル) であることに注意してください。

5.5 その他の例

- `demo_refine.py` スクリプトは、端部がクランプされた単純な 2 次元または 3 次元バーを示します。応力が特異な領域 (クランプ領域と Neumann 境界の間の遷移) でより良い近似を得るために適合リファインメントを用いました。
- `demo_nonlinear_elasticity.py` スクリプトは、曲げ捻りの 3 次元バーを示しています。これは、変形が多数のステップで適用されるため、準静的な問題です。各ステップで非線形 (有限変形) 弾性問題を解きます。
- `demo_stokes_3D_tank.py` スクリプトはタンク内の Stokes(粘性流体) 問題を示します。`demo_stokes_3D_tank_draw.py` はメッシュスライスとストリームラインを使用して、解の適切なプロットを描画する方法を示します。`demo_stokes_3D_tank_alt.py` は古い例で、廃止された `gf_solve` 関数を使っています。
- `demo_bilaplacian.py` スクリプトは、*GetFEM++* の例である `tests/bilaplacian.cc` を改作したものです。bilaplacian(か Kirchhoff-Love のプレートモデル) を正方形で求解します。
- `demo_plasticity.py` スクリプトは、*GetFEM++* の例である `tests/plasticity.cc` を応用したものです。2 次元または 3 次元バーは多数のステップで曲げられ、材料の塑性が考慮されます (材料の Von Mises が所定の閾値を超えたときに塑化が起こります)。
- `demo_wave2D.py` は高次の幾何学的変換と高次の FEM を持つ 2 次元スカラー波動方程式の例 (円柱による平面波の回折) です。

第 6 章

ハウツー

6.1 gmsh のメッシュをインポート

次のようにファイル *quad.geo* にパラメータ化されたメッシュがある場合

```
lc = 0.05 ;

Point(1) = {0,0,0,lc};
Point(2) = {1,0,0,lc};
Point(3) = {1,1,0,lc};
Point(4) = {0,1,0,lc};

Line(5) = {1,2};
Line(6) = {2,3};
Line(7) = {3,4};
Line(8) = {4,1};

Line Loop(9) = {5,6,7,8};
Plane Surface(10) = {9};

Physical Line(101) = {7};
Physical Line(102) = {5};
Physical Line(103) = {8};
Physical Line(104) = {6};

Physical Surface(201) = {10};
```

次のように実行します。

```
$ gmsh -2 quad.geo
```

ファイル *quad.msh* が作成され、メッシュとその領域のエンコーディングが格納されます。このファイル (*quad.msh*) を getfem にインポートできます。

```
import getfem as gf

m = gf.Mesh('import','gmsh','quad.msh')
print m.regions()
```

2 番目のコマンドでは、 *regions ids* が表示されます。メッシュをインポートすると、次のような警告が表示される場合があります。

```
Level 3 Warning in getfem_import.cc, line 137:
  All regions must have different number!
```

これは、 *Gmsh .geo file* で行われるメッシュのパラメータ化で、各領域に **異なる** 番号を割り当ててくださいという意味です。 問題が発生するのは、 *Gmsh* では "領域の種類" が異なるためです。例えば *Gmsh* では "物理面 (200)" と "物理線 (200)" が共存できます。 *GetFEM++* には "領域の種類" が 1 つしかないため、この状態にはなりません。

第 7 章

APIリファレンス

このドキュメントは完備されたものではありません。使用される構造体のアルゴリズムと概念についてのより詳細な説明は、特に user documentation を参照してください。

7.1 ContStruct

class ContStruct(**args*)

GeFEM ContStruct オブジェクト

このオブジェクトは、モデルの解の分岐の numerical continuation(continuation の詳細については、GetFEM++ ユーザーマニュアルを参照) で使用されるパラメータとデータを格納するために使用されます。

ContStruct オブジェクトの汎用的なコンストラクタ

- `S = ContStruct(Model md, string dataname_parameter[, string dataname_init, string dataname_final, string dataname_current], scalar sc_fac[, ...])` 変数 *dataname_parameter* は *md* で与えられるモデルをパラメータ化します。ベクタデータムを介してパラメータ化が行われる場合、 *dataname_init* および *dataname_final* は、パラメータ化を決定するこのデータムの 2 つの与えられた値を格納する必要があり、 *dataname_current* は、このデータムの実際の値を提供します。 *sc_fac* は、continuation で使用される加重ノルムに含まれるスケール係数です。

  その他のオプション

  - 'lsolver', string SOLVER_NAME 組み込まれた線形システムに使用するソルバーの名前 (デフォルト値は 'auto' で、getfem が自分で選択できるようになっています) です。有効な値は 'superlu', 'mumps' (サポートされている場合), 'cg/ildlt', 'gmres/ilu' および 'gmres/ilut' です。

- 'h_init', scalar HIN　初期ステップサイズ (デフォルト 値は 1e-2 です)
- 'h_max', scalar HMAX　最大ステップサイズ (デフォルト 値は 1e-1 です)
- 'h_min', scalar HMIN　最小ステップサイズ (デフォルト 値は 1e-5 です)
- 'h_inc', scalar HINC　ステップサイズを拡大する係数 (デフォルト 値は 1.3 です)
- 'h_dec', scalar HDEC　ステップサイズを減少させる係数 (デフォルト 値は 0.5 です)
- 'max_iter', int MIT　修正で許容される最大反復回数 (デフォルト 値は 10 です)
- 'thr_iter', int TIT　ステップサイズ拡大補正の繰り返し回数閾値 (デフォルト 値は 4 です)
- 'max_res', scalar RES　解析曲線上の新しい点の目標残差値 (デフォルト 値は 1e-6 です)
- 'max_diff', scalar DIFF　連続する 2 点の収束判定基準 (デフォルト 値は 1e-6 です) を決定します。
- 'min_cos', scalar MCOS　古い点と新しい点の解曲線までの接線間の角度の余弦の最小値 (デフォルト 値は 0.9 です)
- 'max_res_solve', scalar RES_SOLVE　解くべき線形系の目標残差値 (デフォルト 値は 1e-8 です)
- 'singularities', int SING　特異点 (1 は極限点、2 は分岐点、2 は特殊な分岐技術が必要な点) の検出と処理のためのツールをアクティブにします
- 'non-smooth'　平滑化されていない問題に特殊な方法を使用できるかどうかを判定します
- 'delta_max', scalar DMAX　異なる平滑部 (デフォルト 値は 0.005 です) に属する二つの拡張 Jacobian の凸包結合の試験関数を評価するための分割の最大サイズ
- 'delta_min', scalar DMIN　凸包結合の試験関数を評価するための最小分割サイズ (デフォルト 値は 0.00012 です)
- 'thr_var', scalar TVAR　分割をリファインするための閾値ばらつき (デフォルト 値は 0.02 です)
- 'nb_dir', int NDIR　新しい片側分岐の位置 (デフォルト 値は 40 です) で新しい正接予測を検索する際の、一対の参照ベクトルの線形結合の総数
- 'nb_span', int NSPAN　線形結合を形成する基準ベクトルの結合の総数 (デフォルト 値は 1 です)

– 'noisy' または 'very_noisy' continuation プロセス中に (残差値等の) 詳細情報をどのように表示するかを決定します。

Moore_Penrose_continuation(*solution*, *parameter*, *tangent_sol*, *tangent_par*, *h*)
Moore-Penrose continuation の 1 つのステップを計算します。 *solution* と *parameter* で与えられる点、 *tangent_sol* と *tangent_par* で与えられる接線、ステップサイズ *h* をとります。解析曲線上の新しい点、対応する接線、次のステップのステップサイズ、および必要に応じて現在のステップサイズを返します。返されたステップサイズが 0 の場合、continuation は失敗しています。オプションで、検出された特異点のタイプを返します。注:最後に、新しい点をモデルに保存する必要はありません。

bifurcation_test_function()
分岐試験関数の最後の値を返し、微分可能性の異なるサブドメイン間を通過する場合は、計算されたグラフ全体を返します。

char()
ContStruct の (ユニークな) 文字列表現を出力します。

これは、2 つの異なる ContStruct オブジェクト間の比較を実行するために使用します。この機能は完成予定です。

compute_tangent(*solution*, *parameter*, *tangent_sol*, *tangent_par*)
更新された接線を計算して返します。

display()
ContStruct オブジェクトの概要を表示します。

init_Moore_Penrose_continuation(*solution*, *parameter*, *init_dir*)
Moore-Penrose continuation を初期化します。 *solution* と *parameter* で指定されるポイントでの解曲線の単位接線と、 continuation の初期ステップサイズを返します。パラメータを基準に計算された正接の方向は *init_dir* の符号によって決まります。

init_step_size()
continuation のための初期ステップサイズを返します。

max_step_size()
continuation の最大ステップサイズを返します。

min_step_size()
continuation の最小ステップサイズを返します。

non_smooth_bifurcation_test(*solution1*, *parameter1*, *tangent_sol1*, *tangent_par1*, *solution2*, *parameter2*, *tangent_sol2*, *tangent_par2*)
tangent_sol1 および *tangent_par1* で指定されたタンジェントを持つ *solution1* および

parameter1 で指定されたポイントと、 *tangent_sol2* および *tangent_par2* で指定されたタンジェントを持つ *solution2* および *parameter2* で指定されたポイントとの間の、滑らかでない分岐ポイントをテストします。

sing_data()
ContStruct オブジェクトに格納されている特異点 (*X*, *gamma*) と、そこから派生するすべての配置済みの解分岐に接線の配列 (*T_X*, *T_gamma*) を返します。

step_size_decrement()
continuation のステップサイズの減少率を返します。

step_size_increment()
continuation のステップサイズの増分比を返します。

7.2 CvStruct

class CvStruct(**args*)
GeFEM CvStruct オブジェクト

CvStruct オブジェクトの汎用的なコンストラクタ

basic_structure()
最も単純な凸包構造を取得します。

たとえば、6 節点三角形の 'basic structure' は標準の 3 節点三角形です。

char()
CvStruct の文字列記述を出力します。

dim()
凸包構造の次数を取得します。

display()
CvStruct オブジェクトの概要が表示されます。

face(*F*)
面 *F* の凸包構造を返します。

facepts(*F*)
面 *F* の点インデックスのリストを返します。

nbpts()
凸包構造のポイント数を取得します。

7.3 Eltm

class Eltm(**args*)

GeFEM Eltm オブジェクト

このオブジェクト は、基本行列のタイプを 表し ます。これら の行列の数値を 取得する には、MeshIm.eltm() を 参照して ください。

非常に特殊なアセンブリ が必要な場合や、基本的な行列の内容をチェックしたい場合には、この関数が便利です。しかし、汎用アセンブリ 関数 gf_asm(...) がほとんどのニーズに適合するはずです。

Eltm オブジェクト の汎用的なコンスト ラクタ

- `E = Eltm('base', Fem FEM)` は有限要素法 *FEM* を 使って 要素の形状関数を 積分する 記述子を 返し ます。
- `E = Eltm('grad', Fem FEM)` は有限要素法 *FEM* を 使って 要素上の形状関数の勾配を 積分するための記述子を 返し ます。
- `E = Eltm('hessian', Fem FEM)` は有限要素法を 使って 要素上の形状関数の Hessian 積分を 表す記述子を 返し ます。
- `E = Eltm('normal')` は凸包面の単位法線の記述子を 返し ます。
- `E = Eltm('grad_geotrans')` は幾何変換の勾配行列への記述子を 返し ます。
- `E = Eltm('grad_geotrans_inv')` は幾何変換の勾配行列の逆行列の記述子を 返し ます (これはめったに使用されません)。
- `E = Eltm('product', Eltm A, Eltm B)` は基本行列 A と B のテンソル積を 積分するための記述子を 返し ます。

7.4 Fem

class Fem(**args*)

GeFEM Fem オブジェクト

このオブジェクト は、参照要素の有限要素法を 表し ます。

Fem オブジェクト の汎用的なコンスト ラクタです。

- `F = Fem('interpolated_fem', MeshFem mf_source, MeshIm mim_target, [ivec blocked_dofs[, bool caching]])` 別 の MeshFem から 補間された 特別な Fem を 構築し ます。

この特殊な有限要素を使用すると、このメッシュで使用される積分法 *mim_target* が指定されている場合に、別のメッシュ上の所定の MeshFem *mf_source* を補間できます。

この有限要素は、キャッシュが使用されているかどうかに応じて、非常に低速であったり、大量のメモリを消費したりする可能性があることに注意してください。デフォルトでは *caching* は True です。

- `F = Fem('projected_fem', MeshFem mf_source, MeshIm mim_target, int rg_source, int rg_target[, ivec blocked_dofs[, bool caching]])` 別の MeshFem から補間された特別な Fem を構築します。

  この特殊な有限要素を使用すると、このメッシュで使用される積分法 *mim_target* が指定されている場合に、別のメッシュ上の所定の MeshFem *mf_source* を補間できます。

  この有限要素は、キャッシュが使用されているかどうかに応じて、非常に低速であったり、大量のメモリを消費したりする可能性があることに注意してください。デフォルトでは *caching* は True です。

- `F = Fem(string fem_name)` *fem_name* には、有限要素法の記述を入力する必要があります。詳細は getfem++ のマニュアル (特に有限要素法と積分法の記述) を参照してください。その一部を次に示します。

  - FEM_PK(n,k) : 次元 *n* のシンプレックス上の古典的 Lagrange 要素 Pk。
  - FEM_PK_DISCONTINUOUS(n,k[,alpha]) : 次元 *n* のシンプレックス上の不連続 Lagrange 要素 Pk。
  - FEM_QK(n,k) : 四辺形、六面体などの古典的な Lagrange 要素 Qk。
  - FEM_QK_DISCONTINUOUS(n,k[,alpha]) : 四辺形、六面体などの不適合 Lagrange 要素 Qk。
  - FEM_Q2_INCOMPLETE (n): 8 および 20 自由度 (セレンディピティ Quad8 と Hexa20 の要素) の不完全 Q2 要素。
  - FEM_PK_PRISM(n,k) : 次元 *n* のプリズム上の古典的 Lagrange 要素 Pk。
  - FEM_PK_PRISM_DISCONTINUOUS(n,k[,alpha]) : プリズム上の古典的な不連続 Lagrange 要素 Pk。
  - FEM_PK_WITH_CUBIC_BUBBLE(n,k) : 追加の体積気泡関数を持つシンプレックス上の従来の Lagrange 要素 Pk。
  - FEM_P1_NONCONFORMING : 三角形の不適合 P1 法。

- FEM_P1_BUBBLE_FACE (n):面 0 に追加の気泡関数を 持つシンプレックス上の P1 法。
- FEM_P1_BUBBLE_FACE_LAG : 面 0 に追加の Lagrange 自由度を 持つシンプレックス上の P1 法。
- FEM_PK_HIERARCHICAL(n,k) : 階層基準の PK 要素。
- FEM_QK_HIERARCHICAL(n,k) : 階層基準の QK 要素
- FEM_PK_PRISM_HIERARCHICAL(n,k) : 階層基準のプリズム上の PK 要素。
- FEM_STRUCTURED_COMPOSITE(Fem f,k) : *k* 分割されたグリッド 上の複合 Fem *f* 。
- FEM_PK_HIERARCHICAL_COMPOSITE(n,k,s) : 細分割および階層ベースを 使用したグリッド 上の Pk 複合要素。
- FEM_PK_FULL_HIERARCHICAL_COMPOSITE(n,k,s) : *s* サブ分割および次数と サブ分割の両方に基づく 階層ベースの Pk 複合要素。
- FEM_PRODUCT(A,B) : 2 つの多項式要素のテンソル積。
- FEM_HERMITE(n) : 次元 *n = 1, 2, 3* の単体上の Hermite 要素 P3。
- FEM_ARGYRIS : 三角形の Argyris 要素 P5。
- FEM_HCT_TRIANGLE : 三角形の Hsieh-Clouh-Tocher 要素 (C1 の複合 P3 要素) は IM_HCT_COMPOSITE() 積分法で使用します。
- FEM_QUADC1_COMPOSITE : 四角形要素、複合 P3 要素および C1(16 自由度)。
- FEM_REDUCED_QUADC1_COMPOSITE : 四辺形要素、複合 P3 要素および C1(12 自由度)。
- FEM_RT0(n) : 次元 *n* のシンプレックス上の次数 0 の Raviart-Thomas 要素。
- FEM_NEDELEC(n) : 次元 *n* のシンプレックス上の次数 0 の Nedelec エッジ要素。

もちろん、選択した有限要素法が幾何変換と 互換性があることを 確認する 必要があります。つまり、四角形では Pk 有限要素法は意味を 持ちません。

base_value(*p*)

点 *p* における FEM のすべての基底関数を 評価します。

p は参照凸包の中にあるはずです!

char()

Fem の (ユニークな) 文字列表現を 出力します。

これを使用して、2 つの異なる Fem オブジェクト の比較ができます。

dim()
Fem の次数 (参照凸包の次数) を返します。

display()
Fem オブジェクト の概要が表示されます。

estimated_degree()
Fem の多項式次数の推定値を返します。

これは多項式ではない有限要素法に対する推定です。

grad_base_value(*p*)
点 p における Fem のすべての基本関数の勾配を評価します。

p は参照凸包の中にあるはずです!

hess_base_value(*p*)
点 p における Fem のすべての基底関数の Hessian を評価します。

p は参照凸包の中にあるはずです!

index_of_global_dof(*cv*)
補間された有限要素法など、特殊な有限要素法の全体自由度のインデックスを返します。

is_equivalent()
Fem が等価でない場合は 0 を返します。

等価 Fem は参照凸包上で評価されます。これはほとんどの古典的な Fem の場合でそうです。

is_lagrange()
Fem が Lagrange 型でない場合は 0 を返します。

is_polynomial()
基底関数が多項式でない場合は 0 を返します。

nbdof(*cv=None*)
Fem の自由度を返します。

特定の Fem(例えば 'interpolated_fem') の中には、その結果を得るために凸包番号 *cv* を必要とするものがあります。ほとんどの場合、この凸包番号は省略できます。

poly_str()
参照凸包の基底関数の多項式を返します。

結果は文字列のタプルとして表現されます。もちろん、これは多項式でない Fem では失敗します。

pts(*cv=None*)
基準エレメント上の自由度の位置を取得します。

ある特定の Fem は、その結果を与えるために凸の数 *cv* を必要とします (例えば 'interpolated_fem')。ほとんどの場合、この凸の数は省略できます。

target_dim()
ターゲット空間の次元を返します。

ベクトル Fem を除き、ターゲット空間の次元は通常 1 です。

7.5 GeoTrans

class GeoTrans(**args*)
GeFEM GeoTrans オブジェクト

幾何変換は、凸包によるカスタムメッシュ凸包を構築する場合に使用する必要があります (Mesh の add_convex() 関数を参照してください)。これにより、凸包の種類 (三角形、六面体、プリズムなど) も定義されます。

GeoTrans オブジェクトの汎用的なコンストラクタ

- `GT = GeoTrans(string name)` name 引数は幾何変換の仕様を文字列として含んでいます。
  - GT_PK(n,k) : 次元 *n* 、次数 *k* のシンプレックス上の変換。
  - GT_QK(n,k) : 平行六面体、次元 *n* 、次数の *k* の変換。
  - GT_PRISM(n,k) : プリズム上、次元 *n* 、次数 *k* の変換。
  - GT_PRODUCT(A,B) : 2 つの変換のテンソル積。
  - GT_LINEAR_PRODUCT(GeoTrans gt1,GeoTrans gt2) : 2 つの変換の線形テンソル積

char()
GeoTrans の (ユニークな) 文字列表現を出力します。

これを使用して、2 つの異なる GeoTrans オブジェクト間の比較を実行できます。

dim()
GeoTrans の次元が得られます。

これはソース空間の次元、すなわち参照凸包の次元です。

display()
GeoTrans オブジェクト の簡単な概要が表示されます。

is_linear()
GeoTrans が線形でない場合は 0 を返します。

nbpts()
GeoTrans のポイント 数を 返します。

normals()
GeoTrans の参照凸包の各面の法線を 取得します。

法線は出力行列の列に 格納されます。

pts()
GeoTrans の参照凸包点を 返します。

ポイント は出力行列の列に 格納されます。

transform(*G*, *Pr*)
GeoTrans を 一連のポイント に 適用します。

G は実際の凸包の頂点の集合であり、 *Pr* は (基準凸包の) 変換される 点の集合です。実際の凸包の対応する 点の集合を 返します。

7.6 GlobalFunction

class GlobalFunction(**args*)
GeFEM GlobalFunction オブジェクト

Global function オブジェクト は次の 3 つの関数で表されます。

- 関数 *val* 。
- 勾配関数 *grad* 。
- Hessian 関数 *hess* 。

このタイプの関数は、ローカルおよびグローバルな enrichment 関数として使用されます。グローバル関数 Hessian は (4 次微分問題のみの) オプションのパラメータです。

GlobalFunction オブジェクト の汎用的なコンストラクタ

- `GF = GlobalFunction('cutoff', int fn, scalar r, scalar r1, scalar r0)` はグローバルなカット オフ関数を 作成します。

- `GF = GlobalFunction('crack', int fn)` 亀裂をモデル化するためのグローバルな近端漸近関数を作成します。
- `GF = GlobalFunction('parser', string val[, string grad[, string hess]])` 文字列 *val*, *grad* および *hess* からグローバル関数を作成します。この関数は、汎用アセンブリ言語の微分を使用することができます... 完成予定です。
- `GF = GlobalFunction('product', GlobalFunction F, GlobalFunction G)` 2 つのグローバル関数の積を作成します。
- `GF = GlobalFunction('add', GlobalFunction gf1, GlobalFunction gf2)` 2 つのグローバル関数を加算します。

char()
GlobalFunction の (ユニークな) 文字列表示を出力します。

これを使用して、2 つの異なる GlobalFunction オブジェクト間の比較を実行できます。この機能は完成予定です。

display()
GlobalFunction オブジェクトの簡単な概要を表示します。

grad(*PTs*)
Return *grad* function evaluation in *PTs* (column points).

返される *GRADs* の各列は [Gx,Gy] の形式になっています。

hess(*PTs*)
Return *hess* function evaluation in *PTs* (column points).

返される *HESSs* の各列は [Hxx,Hxy,Hyx,Hyy] という形式になっています。

val(*PTs*)
Return *val* function evaluation in *PTs* (column points).

7.7 Integ

class Integ(**args*)
GeFEM Integ オブジェクト

凸包のさまざまな積分法のハンドルを取得するための汎用オブジェクトです (要素行列が構築されるときに使われます)。

Integer オブジェクトの汎用的なコンストラクタ

- `I = Integ(string method)` getfem++(完全なリファレンスについては、有限要素および積分法の説明を参照してください) で定義されている積分法のリストを以下に示します。
  - IM_EXACT_SIMPLEX(n) : シンプレックス完全積分 (線形幾何変換と PK 有限要素法)。
  - IM_PRODUCT(A,B) : 2 つの積分法の積。
  - IM_EXACT_PARALLELEPIPED(n) : 平行六面体の完全積分。
  - IM_EXACT_PRISM(n) : プリズムの完全積分。
  - IM_GAUSS1D(k) : セグメントの Gauss 法、次数 *k=1,3,...,99* 。
  - IM_NC(n,k) : 次数 *k* のシンプレックス上の近似積分を行う Newton-Cotes。
  - IM_NC_PARALLELEPIPED(n,k) : 平行六面体上の Newton-Cotes 積分の積。
  - IM_NC_PRISM(n,k) : 角柱の Newton-Cotes 積分の積。
  - IM_GAUSS_PARALLELEPIPED(n,k) : 平行六面体上の Gauss1 次積分の積。
  - IM_TRIANGLE(k) : 三角形の Gauss 法 *k=1,3,5,6,7,8,9,10,13,17,19* 。
  - IM_QUAD(k) : 四辺形上の Gauss 法 *k=2,3,5, ...,17* 。QK 有限要素法には IM_GAUSS_PARALLELEPIPED が好まれます。
  - IM_TETRAHEDRON(k) : 四面体の Gauss 法 *k=1,2,3,5,6* または *8* 。
  - IM_SIMPLEX4D(3) : 4 次元シンプレックス上の Gauss 法。
  - IM_STRUCTURED_COMPOSITE(im,k) : *k* 分割されたグリッド上の複合法です。
  - IM_HCT_COMPOSITE(im) : HCT 複合有限要素に適した複合積分。

  例:
  - I = Integ('IM_PRODUCT(IM_GAUSS1D(5),IM_GAUSS1D(5))')

  は以下と同じです。
  - I = Integ('IM_GAUSS_PARALLELEPIPED(2,5)')

  'exact integration' は線形の幾何変換にのみ適用され、非常に遅く、高次の有限要素法に対しては数値的安定性の問題があるため、一般的には避けるべきであることに留意してください。

`char`()
積分法の (ユニークな) 文字列表現を出力します。

これは、2 つの異なる Integ オブジェクト 間の比較に使用できます。

coeffs()
各積分点に関連付けられた係数を返します。

近似法の場合のみで、完全積分法では意味はありません!

dim()
積分法の参照凸包の次元を返します。

display()
Integ オブジェクト の簡単な概要を表示します。

face_coeffs(*F*)
面の各積分に関連付けられた係数を返します。

近似法の場合のみで、完全積分法では意味はありません!

face_pts(*F*)
面の積分点のリストを返します。

近似法の場合のみで、完全積分法では意味はありません!

is_exact()
積分が近似の場合は 0 を返します。

nbpts()
積分点の総数を返します。

参照凸包の各面の体積積分点と面積分点をカウントします。

近似法の場合のみで、完全積分法では意味はありません!

pts()
積分点のリストを返します

近似法の場合のみで、完全積分法では意味はありません!

7.8 LevelSet

class LevelSet(**args*)
GeFEM LevelSet オブジェクト

level-set オブジェクト は、プライマリ level-set と、オプションで破壊を表すために使用される セカンダリ level-set(p(x) がプライマリ level-set 関数で、 s(x) がセカンダリ level-set の場合、

亀裂は次の式で定義されます。 $p(x) = 0$ と $s(x) \leq 0$:セカンダリの役割は、亀裂の前面/先端を決定することです) によって表されます。

注:

次に示すすべてのツールでは、パッケージ qhull がシステムにインストールされている必要があります。このパッケージは広く入手可能です。任意の次元における凸包と Delaunay 三角形分割を計算します。

LevelSet オブジェクトの汎用的なコンストラクタです。

- `LS = LevelSet(Mesh m, int d[, string 'ws'| string f1[, string f2 | string 'ws']])` メッシュ上に LevelSet オブジェクトを作成します。このオブジェクトは一次関数 (およびオプションの 2 次関数の両方) で表現され、次数 d の MeshFem で定義されます。

  (セカンダリの) *ws* が設定されている場合、この levelset はプライマリ関数とセカンダリ関数で表されます。 *f1* が設定されている場合、基本関数は (高水準汎用アセンブリ言語の構文を使用した) 表現で定義されます。 *f2* が設定されている場合、この levelset は、これらの式で定義されるプライマリ関数とセカンダリ関数で表されます。

char()
LevelSet の (一意な) 文字列表現を出力します。

これを使用して、2 つの異なる LevelSet オブジェクト間の比較を実行できます。この機能は完成予定です。

degree()
lagrange 表現の次数を返します。

display()
LevelSet の簡単な概要を表示します。

memsize()
level-set で使用されるメモリーの量 (バイト単位) を返します。

mf()
MeshFem オブジェクトの参照を返します。

set_values(**args*)
概要: LevelSet.set_values(self, {mat v1|string func_1}[, mat v2|string func_2])

level-set 関数の自由度のベクトルの値を設定します。

1 次関数を *v1* 自由度 (または表現 *func_1*) のベクトルで設定し、(存在する場合)2 次関数を *v2* 自由度 (または表現 *func_2*) のベクトルで設定します。

simplify(*eps=0.01*)

オプションでパラメータ *eps* を使用して、level-set の自由度を単純化します。

values(*nls*)

nls 関数の自由度のベクトルを返します。

nls が 0 の場合、このメソッドはプライマリ level-set 関数の自由度のベクトルを返します。*nls* が 1 の場合、このメソッドは (存在する場合)2 次 level-set 関数の自由度のベクトルを返します。

7.9 Mesh

class Mesh(**args*)

GeFEM Mesh オブジェクト

このオブジェクトは、異なる次元の要素を混在させても、任意の次元の任意の要素を格納できます。

Mesh オブジェクトの汎用的なコンストラクタ

- `M = Mesh('empty', int dim)` 新しい空のメッシュを作成します。
- `M = Mesh('cartesian', vec X[, vec Y[, vec Z,..]])` 四角形や立方体などの規則的なメッシュを素早く作成します。
- `M = Mesh('pyramidal', vec X[, vec Y[, vec Z,..]])` ピラミッドなどの規則的なメッシュを素早く作成します。
- `M = Mesh('cartesian Q1', vec X, vec Y[, vec Z,..])` Q1 要素で四角形や立方体などの規則的なメッシュを素早く作成します。
- `M = Mesh('triangles grid', vec X, vec Y)` 三角形の規則的なメッシュを素早く作成します。

  これは非常に限定された関数であり、何らかの理由で廃止された関数です (`Mesh('ptND')`, `Mesh('regular simplices')` と `Mesh('cartesian')` も参照)。
- `M = Mesh('regular simplices', vec X[, vec Y[, vec Z,... ]][...])` 単純なメッシュ (三角形、四面体など) による n 元の平行四面体を作成します。

  オプションの次数を使用して、非線形幾何変換でメッシュを構築できます。

- `M = Mesh('curved', Mesh m, vec F)` n 次元のメッシュ *m* から曲 (n+1) 次元のメッシュを構築します。

  新しいメッシュの点には、ベクトル *F* によって与えられる追加の座標が 1 つあります。これを使用して、シェルのメッシュを取得できます。 *m* は MeshFem オブジェクトも指定可能です。その場合はリンクされたメッシュが使用されます。

- `M = Mesh('prismatic', Mesh m, int nl[, int degree])` Mesh *M* から角柱 Mesh *M* を押し出します。

  付加次元には、 `0` から `1` に分布する要素の *nl* 層があります。オプションのパラメータ *degree* にデフォルト値 `1` より大きい値を指定すると、対応する次数の非線形変換が押し出し方向で考慮されます。

- `M = Mesh('pt2D', mat P, imat T[, int n])` 2 次元三角形分割からメッシュを構築します。

  P の各列は点座標を含み、 *T* の各列は三角形の点インデックスを含みます。 *n* はオプションであり、領域番号です。 *n* が指定された場合、領域番号 *n* のみが変換されます (この場合、 *T* には 4 つの行があり、4 行目には領域番号が入ります)。

- `M = Mesh('ptND', mat P, imat T)` n 次元 "triangulation" からメッシュを構築します。

  T で与えられた三角形分割と *P* で与えられた点のリストからシンプレックスメッシュを構築する 'pt2D' と同様の関数。メッシュの次元は *P* の行数であり、シンプレックスの次元は *T* の行数です。

- `M = Mesh('load', string filename)` getfem++ ASCII メッシュファイルからメッシュをロードします。

  `Mesh.save(string filename)` も参照してください。

- `M = Mesh('from string', string s)` string description からメッシュをロードします。

  例えば、 `Mesh.char()` が返す文字列。

- `M = Mesh('import', string format, string filename)` メッシュをインポートします。

  format には次のいずれかを指定できます。

  – 'gmsh' は *Gmsh* で作成されたメッシュを表します。

  – 'gid' は *GiD* で作成されたメッシュを表します。

  – 'cdb' は *ANSYS* で作成されたメッシュを表します。

– 'am_fmt' は *EMC2* で作成されたメッシュを表します。

- `M = Mesh('clone', Mesh m2)` メッシュのコピーを作成します。
- `M = Mesh('generate', MesherObject mo, scalar h[, int K = 1[, mat vertices]])` *mo* は Getfem の実験的幾何メッシャーを指します。作成したメッシュの整合性は制御してください。配列 *vertices* にアプリオリに頂点を追加して、メッシュの次元 n を `n`、点の数を `m` として、サイズを `n x m` とすることで、メッシャーを支援できます。 *h* は要素のおおよその直径です。 *K* はメッシュの次数 (>1 曲線境界の場合) です。メッシャは要素の品質を最適化しようとします。この操作には時間がかかる場合があります。メッシュの生成が失敗した場合は、使用されたランダムプロシージャのために必ずしも同じ結果が得られないため、再度実行できます。メッシュ生成によってコンソールに送信されるメッセージは、 *gf_util('trace level', 2)* を使って非アクティブにすることができる。より詳しい情報は *gf_util('trace level', 4)* で得ることができる。ジオメトリを記述するためにジオメトリプリミティブを操作するには、 `MesherObject` を参照してください。

add_convex (*GT*, *PTS*)

メッシュに新しい凸包を追加します。

凸包 (三角形、プリズム、...) は *GT* (GeoTrans('...') で取得されます) によって与えられ、その点は *PTS* の列によって与えられます。返される *CVID* には凸包の #id が含まれます。 *PTS* は複数の凸包 (または Fortran の次数に従って正しく形成された 2 次元配列) を挿入するための 3 次元配列の場合もあります。

add_point (*PTS*)

メッシュに新しいポイントを挿入し、その #id を返します。

PTS は `nxm` 行列でなければなりません。ここで `n` はメッシュの次元、 `m` はメッシュに追加される点の数です。出力では、 *PID* にこれらの新しいポイントのポイント番号 ID が含まれます。

注: (`1e-8` 程度の小さな許容範囲で) 既にメッシュの一部になっているポイントがある場合、それらのポイントは再挿入されず、 *PIDs* には以前に割り当てられた #id のポイントが含まれます。

adjacent_face (*cvid*, *fid*)

存在する場合、隣接する要素の凸面を返します。凸包が面 `f` (例えば、3 次元で要素を抑制することを考える) に対して相対的に隣接する面を複数持つ場合、最初に見つかった面を返します。

all_faces (*CVIDs=None*)

CVID(CVID が省略されている場合、すべてのメッシュで) での面の集合を返します。2 つの隣接する要素が共有する面が 2 回表示されることに注意してください。

boundaries()
非推奨関数です。代わりに 'regions' を使用してください。

boundary()
非推奨関数です。代わりに 'regions' を使用してください。

char()
メッシュの string description を出力します。

convex_area(*CVIDs=None*)
各凸包の面積の推定値を返します。

convex_radius(*CVIDs=None*)
各凸包の半径の推定値を返します。

convexes_in_box(*pmin*, *pmax*)
コーナーポイント *pmin* と *pmax* で定義されたボックス内に完全に収まる凸包集合を返します。

出力される *CVID* は二行行列で、最初の行には凸の#id がリストされ、次の行には面の番号 (凸包のローカル番号) がリストされます。 *CVID* が指定された場合、 *CVID* にリストされた#id によって定義される凸包集合の境界の一部を返します。

curved_edges(*N*, *CVLST=None*)
[廃止される関数! 将来のリリースで削除される予定です]

曲線要素用に設計された Mesh.edges() のより洗練されたバージョン。これは (曲がった) エッジの N(N>=2) 点を返します。N==2 の場合、これは Mesh.edges() と同等です。点は常にメッシュの一部ではないため、 [mesh_dim x 2 x nb_edges] 配列である配列 E で、点の番号の代わりに座標が返されます。オプションの出力引数 C を指定すると、各エッジに関連付けられた凸包の番号が含まれます。

cvid()
すべての凸包#id のリストを返します。

番号付けは、特にメッシュからいくつかのポイントが削除された場合は、0 から Mesh.nbcvs()-1 まで連続していないことに注意してください。 Mesh.optimize_structure() を使用すると、連続した番号付けが行われます。

cvid_from_pid(*PIDs*, *share=False*)
PIDs で指定されたポイント #ids に関連する凸包 #ids を検索します。

share=False の場合は、頂点#ids が *PIDs* にある凸包を検索します。 *share=True* の場合は、 *PIDs* で指定されたポイント #ids を共有する凸包 #ids を検索します。 *CVIDs* は (空である可能性がある) ベクトルです。

cvstruct(*CVIDs=None*)
凸包構造の配列を返します。

CVIDs が指定されていない場合、すべての凸包が考慮されます。各凸包構造は *S* に 1 つずつ記載されており、 *CV2S* は *CVIDs* の凸包のインデックスを *S* の構造インデックスにマップしています。

del_convex(*CVIDs*)
メッシュから 1 つまたは複数の凸包を削除します。

CVIDs には 'add convex' コマンドで返されるような凸包 #id を含める必要があります。

del_convex_of_dim(*DIMs*)
DIMs に表示されている次元の凸包をすべて削除します。

例えば `Mesh.del_convex_of_dim([1,2])` はすべての線区分、三角形、四角形を削除します。

del_point(*PIDs*)
メッシュから 1 つまたは複数のポイントを削除します。

PIDs は 'add point' コマンドで返されるポイントである #ids などを指定する必要があります。

delete_boundary(*rnum*, *CVFIDs*)
非推奨関数です。代わりに 'delete region' を使用してください。

delete_region(*RIDs*)
RIDs にリストされている領域 #id を削除します。

dim()
メッシュの次元を取得します (2 次元メッシュに対して 2 など)。

display()
Mesh オブジェクトの簡単な概要を表示します。

edges(*CVLST=None*, **args*)
概要: [E,C] = Mesh.edges(self [, CVLST][, 'merge'])

[廃止される関数! 将来のリリースで削除される予定です]

行ベクトル CVLST にリストされている凸包のメッシュ M のエッジのリストを返します。E はポイントインデックスを含む 2 x nb_edges 行列です。CVLST を省略すると、すべての凸包のエッジが返されます。CVLST に 2 つの行がある場合、最初の行には凸包の数が含まれ、2 番目の行には面の数が返されます。 'merge' が指定されている場合、凸包のすべての共通エッジが単一エッジにマージされます。オプションの出力引数 C を指定すると、各エッジに関連付けられた凸包の番号が含まれます。

export_to_dx(*filename*, **args*)

概要: Mesh.export_to_dx(self, string filename, ... [,'ascii'][,'append'][,'as',string name,['serie',string serie_name]][,'edges'])

メッシュを OpenDX ファイルに書き出します。

MeshFem.export_to_dx(), Slice.export_to_dx() も参照してください。

export_to_pos(*filename*, *name=None*)

メッシュを POS ファイルに書き出します。

MeshFem.export_to_pos(), Slice.export_to_pos() も参照してください。

export_to_vtk(*filename*, **args*)

概要: Mesh.export_to_vtk(self, string filename, ... [,'ascii'][,'quality'])

メッシュを VTK ファイルに書き出します。

'quality' が指定された場合、各凸包の品質の見積りがファイルに書き込まれます。

See also MeshFem.export_to_vtk(), Slice.export_to_vtk().

extend_region(*rnum*, *CVFIDs*)

領域番号 *rnum* で識別される領域を拡張して、マトリックス *CVFIDs* に与えられた凸包または凸包面あるいはその両方の集合を含めます。 `Mesh.(set region)` も参照。

faces_from_cvid(*CVIDs=None*, **args*)

概要: CVFIDs = Mesh.faces_from_cvid(self[, ivec CVIDs][, 'merge'])

凸包 #id のリストから凸包面のリストを返します。

CVFIDs は 2 行の行列で、最初の行には凸包の#id がリストされ、2 番目の行には面番号 (凸包のローカル番号) がリストされます。 *CVIDs* が指定されていない場合、すべての凸包が対象となります。オプションの引数 'merge' は *CVIDs* の凸包が共有している面を結合します。

faces_from_pid(*PIDs*)

頂点 #ids が *PIDs* にある凸包面を返します。

CVFIDs は 2 行の行列で、最初の行には凸包の#id がリストされ、2 番目の行には面番号 (凸包のローカル番号) がリストされます。凸包面を返すには、その各点を *PIDs* にリストする必要があります。

geotrans(*CVIDs=None*)

幾何変換の配列を返します。

Mesh.cvstruct() も参照してください。

inner_faces(*CVIDs=None*)

CVID の 2 つ以上の要素が共有する面の集合を返します。各面は 1 回だけ表示され、2 つの隣接エレメント の間で任意に選択されます。

max_cvid()

メッシュ内のすべての凸包の最大#id を返します ('max pid' を参照してください。)。

max_pid()

メッシュ内のすべてのポイント の最大 #id を返します ('max cvid' を参照してください)。

memsize()

メッシュが使用するメモリ の量 (バイト 単位) を返します。

merge(*m2*, *tol=None*)

メッシュ *m2* とマージします。

許容半径 *tol* 内の重複ポイント は複製されません。 *m2* が MeshFem オブジェクト の場合は、そのリンクされたメッシュが使用されます。

nbcvs()

メッシュの凸包の数を 取得します。

nbpts()

メッシュのポイント 数を 取得します。

normal_of_face(*cv*, *f*, *nfpt=None*)

面の *nfpt* 点で凸包の *cv* 、面 *f* の法線を 評価します。

nfpt が指定されていない場合、法線は面の各ジオメトリ ノード で評価されます。

normal_of_faces(*CVFIDs*)

凸包の法線を (面の中心で) 評価します。

CVFID は 2 行の行列を 想定しており 、最初の行は凸包の #id をリストし、2 番目の行は面番号 (凸包のローカル番号) をリストしています。

optimize_structure(*with_renumbering=None*)

点および凸包の番号付けをリセットします。

最適化後、ポイント (resp. convexes) には連続して `0` から `Mesh.max_pid()-1` (resp. `Mesh.max_cvid()-1`) まで番号が付けられます。

orphaned_pid()

凸包にリンクされていないポイント #id の検索。

outer_faces(*CVIDs=None*)

2 つの要素で共有されていない面の集合を 返します。

出力される *CVFIDs* は 2 行の行列であり、最初の行は凸包 #id をリストし、二番目の行は面番号 (凸包の局所番号) をリストします。*CVIDs* が指定されていない場合、すべての凸包が考慮され、基本的にメッシュ境界が返されます。 *CVIDs* が指定された場合、 *CVIDs* に #id リストの凸包集合の境界を返します。

outer_faces_in_box(*pmin*, *pmax*, *CVIDs=None*)

2 つの凸包によって共有されておらず、コーナーポイント *pmin* と *pmax* によって定義されるボックス内にある面の集合を返します。

出力される *CVFIDs* は 2 行の行列であり、最初の行は凸包の #ids をリストし、2 番目の行は面番号 (凸包の局所番号) をリストします。 *CVIDs* が指定された場合、 *CVIDs* にリストされた #id によって定義される凸包集合の境界の一部を返します。

outer_faces_with_direction(*v*, *angle*, *CVIDs=None*)

2 つの凸包によって共有されておらず、平均外向きベクトルがベクトル *v* からの角度 *angle* (ラジアン) 内にある面の集合を返します。

出力される *CVFIDs* は 2 行の行列であり、最初の行は凸包の #ids をリストし、2 番目の行は面番号 (凸包の局所番号) をリストします。 *CVIDs* が指定された場合、 *CVIDs* にリストされた #id によって定義される凸包集合の境界の一部を返します。

pid()

メッシュのポイント #id のリストを返します。

番号付けは、0 から Mesh.nbpts()-1 まで連続していないことに注意してください。Mesh.optimize_structure() を使用すると、連続した番号付けが行われます。

pid_from_coords(*PTS*, *radius=0*)

座標が *PTS* にリストされている検索ポイント #id 。

PTS は、点の座標のリストを含む配列です。返される *PIDs* は、 *eps* の範囲内で見つかった各ポイントのポイント #id です。メッシュ内で見つからなかった場合には -1 が返されます。

pid_from_cvid(*CVIDs=None*)

メッシュの各凸包に取り付けられたポイントを返します。

CVIDs (*CVIDs = Mesh.max_cvid()* と同じです) を省略すると、すべての凸包が対象となります。 *IDx* はベクトルであり、 length(IDx) = length(CVIDs)+1 です。 *Pid* は *CVIDs* における各凸包の点の #id の連結リストを含むベクトルです。 *IDx* の各エントリは、 *Pid* の対応する凸包点リストの位置です。したがって、例えば、第 2 凸包の点の #id のリストは Pid[IDx(2):IDx(3)] です。

CVIDs にメッシュに存在しない凸包の#id が含まれている場合、点リストは空になります。

pid_in_cvids(*CVIDs*)

CVIDs にリストされているポイント #id の検索。

PIDs はポイント #id を含むベクトルです。

pid_in_faces(*CVFIDs*)

CVFIDs にリストされているポイント #id の検索。

CVFIDs は 2 行の行列で、最初の行には凸包の #id がリストされ、2 番目の行には面の番号がリストされます。返される *PIDs* は、点#id を含むベクトルです。

pid_in_regions(*RIDs*)

RIDs にリストされているポイント #id の検索。

PIDs はポイント #id を含むベクトルです。

pts(*PIDs=None*)

メッシュのポイントの座標リストを返します。

返されるマトリックスの各列には、各点の座標が含まれます。オプションの引数 *PIDs* が指定された場合、このベクトルに #id がリストされているポイントのみが返されます。それ以外の場合、返されるマトリックスは Mesh.max_pid() 列になります。(メッシュの一部のポイントが破棄され、 Mesh.optimize_structure() が呼び出されていない場合) Mesh.nbpts() より大きい可能性があります。削除された点に対応する列は NaN で埋められます。そのような空のポイントを取り除くには Mesh.pid() を使用します。

pts_from_cvid(*CVIDs=None*)

CVID にリストされているポイントの検索。

CVIDs (*CVIDs* = *Mesh.max_cvid()* と同じです) を省略すると、すべての凸包が考慮されます。 *IDx* はベクトルであり、length(IDx) = length(CVIDs)+1 です。 *Pts* は *CVIDs* における各凸包の点の連結リストを含むベクトルです。 *IDx* の各エントリは *Pts* の対応する凸包ポイントリストの位置です。したがって、例えば、第 2 凸包のポイントのリストは Pts[:,IDx[2]:IDx[3]] となります。

CVIDs にメッシュに存在しない凸包の#id が含まれている場合、点リストは空になります。

quality(*CVIDs=None*)

各凸包の品質の推定値を返します ($0 \leq Q \leq 1$)。

refine(*CVIDs=None*)

メッシュの改善には Bank の方法を使用します。

CVIDs が指定されていない場合、メッシュ全体が洗練されます。このメッシュにリンクされた MeshFem オブジェクトと MeshIm オブジェクトの領域、有限要素法、および積分法は、自動的に洗練されます。

region(*RIDs*)
領域 *RIDs* 上の凸包/面のリストを返します。

CVFIDs は 2 行の行列で、最初の行には凸包の #id がリストされ、2 番目の行には面番号 (凸包のローカル番号) がリストされます。(凸包全体が領域内にある場合は -1 です)。

region_intersect(*r1*, *r2*)
領域番号 *r1* を領域番号 *r2* との交差部分に置き換えます。

region_merge(*r1*, *r2*)
領域番号 *r2* を領域番号 *r1* にマージします。

region_subtract(*r1*, *r2*)
領域番号 *r1* を領域番号 *r2* との差分に置き換えます。

regions()
メッシュに保存されている有効な領域のリストを返します。

save(*filename*)
メッシュオブジェクトを ASCII ファイルに保存します。

このメッシュは Mesh('load', filename) で復元できます。

set_boundary(*rnum*, *CVFIDs*)
非推奨関数です。代わりに 'regions' を使用してください。

set_pts(*PTS*)
メッシュ点の座標を *PTS* で指定された座標に置き換えます。

set_region(*rnum*, *CVFIDs*)
領域番号 *rnum* を、行列 *CVFIDs* に提供されている凸包または凸包面、あるいはその両方、の集合に割り当てます。

CVFIDs の最初の行は凸包の #id を含み、二番目の行は凸包の中に面番号を含みます (または凸包全体として `-1` (領域は通常、凸包面のリストを格納するために使用されますが、凸包面のリストを格納するために使用することもできます)。

ベクトル (または 1 行の行列) が与えられる場合、領域は対応する凸包集合を表します。

transform(*T*)
メッシュの各点にマトリックス *T* を適用します。

なお、 *T* は `NxN ( N = Mesh.dim() )` 行列である必要はありません。したがって、2 次元メッシュを 3 次元メッシュに (相互に) 変換できます。

translate(*V*)
メッシュの各点を *V* から移動します。

triangulated_surface(*Nrefine*, *CVLIST=None*)

[非推奨! は将来のリリースで削除される 予定です]

Mesh.curved_edges() に似た関数です。各面をサブ三角形に分割 (必要に応じて、つまり非線形の場合は幾何変換) して、T (gf_compute('eval on P1 tri mesh') を参照) の座標を返します。

7.10 MeshFem

class MeshFem(**args*)

GetFEM MeshFem オブジェクト

このオブジェクト は、メッシュ全体で定義された有限要素法を表します。

MeshFem オブジェクト の汎用的なコンスト ラクタ

- `MF = MeshFem(Mesh m[, int Qdim1=1[, int Qdim2=1, ...]])` 新しい MeshFem オブジェクトを構築します。

  Qdim パラメータは、有限要素法によって表されるフィールド の次元を 指定します。スカラー場の場合は Qdim1 = 1、サイズ n 以外のベクト ル場の場合は Qdim1 = n 、サイズ mxn の行列場の場合は Qdim1=m、Qdim2=n ... 作成されたオブジェクト のハンド ルを返します。

- `MF = MeshFem('load', string fname[, Mesh m])` ファイルから MeshFem をロードします。

  メッシュ *m* が与えられない場合 (この種類のファイルにはメッシュは保存されません)、ファイル *fname* から 読み込まれ、そのディスクリプタが第 2 の出力引数として返されます。

- `MF = MeshFem('from string', string s[, Mesh m])` string description から MeshFem オブジェクトを 作成します。

  `MeshFem.char()` も参照してください。

- `MF = MeshFem('clone', MeshFem mf)` MeshFem のコピーを 作成します。

- `MF = MeshFem('sum', MeshFem mf1, MeshFem mf2[, MeshFem mf3[, ...]])` 2 つ (かそれ以上の)MeshFem にわたる MeshFem を 作成します。

  すべての MeshFem は同じメッシュを 共有する 必要があります。

  使用後は、 *mf1*, *mf2* の FEM を修正しないでください。

- `MF = MeshFem('product', MeshFem mf1, MeshFem mf2)` は、選択した *mf1* の形状関数のすべての積と、 *mf2* のすべての形状関数のすべての積を網羅する MeshFem を作成します。Xfem enrichment 用に設計されています。

  mf1 と *mf2* は同じメッシュを共有しなければなりません。

  使用後は、 *mf1*, *mf2* の FEM を修正しないでください。

- `MF = MeshFem('levelset', MeshLevelSet mls, MeshFem mf)` MeshLevelSet で定義されたインプリシット サーフェスにコンフォーマルな MeshFem を作成します。

- `MF = MeshFem('global function', Mesh m, LevelSet ls, (GlobalFunction GF1,...)[, int Qdim_m])` 基底関数が *ls* の 2 つのレベルセット関数の iso 値によって定義された座標系でユーザーが指定したグローバル関数である MeshFem を作成します。

- `MF = MeshFem('partial', MeshFem mf, ivec DOFs[, ivec RCVs])` *mf* の自由度のサブセットのみを保持することで、制限された MeshFem を構築します。

  RCVs が与えられた場合、 *RCVs* に列挙されている凸包には FEM は適用されません。

adapt()
MeshFem levelset オブジェクトの場合のみに使用されます。mesh_fem オブジェクトを levelset 関数の変更に適合させます。

basic_dof_from_cv(*CVids*)
CVids にリストされている凸包の自由度を返します。

警告: 自由度は任意の順序で返される可能性があります。アセンブリルーチンではこの関数を使用しないでください。凸包値に関連する自由度をマップする場合は、代わりに 'basic dof from cvid' を使用します。

また、(最初の行の凸包数に関して)2 列目に面番号を表示することで、凸包面上の集合の基本的な自由度のリストを得ることができます。

basic_dof_from_cvid(*CVids=None*)
メッシュの各凸包にアタッチされた自由度を返します。

CVids を省略すると、すべての凸包が考慮されます (*CVids = 1 ... Mesh.max_cvid()* と同じです)。

IDx はベクトルであり、 *length(IDx) = length(CVids)+1* です。 *DOFs* は *CVids* の各凸包の自由度の連結リストを含むベクトルです。 *IDx* の各エントリは *DOFs* における対応する凸包点リストの位置です。従って、例えば、第 2 凸包の点のリストは DOFs[IDx(2):IDx(3)] です。

CVids にメッシュに存在しない凸包の #id が含まれている場合、点リストは空になります。

basic_dof_nodes(*DOFids=None*)

基本自由度の位置を取得します。

指定した *DOFids* (*DOFids* を省略すると、すべての基本自由度が考慮されます) の自由度 #IDs の補間点のリストを返します。

basic_dof_on_region(*Rs*)

Rs にリストされているメッシュ領域のいずれかにある基本自由度 (任意削減前) のリストを返します。

より正確には、この関数は *Rs* (境界領域の場合、一部の節点自由度は正確に境界上に存在しない場合があります。たとえば、 Pk(n,0) の自由度は凸包の中心に存在しますが、基底関数は凸包境界上で null ではありません) に #ids がリストされている領域のいずれかにおいて、サポートが非 NULL である基本的な自由度を返します。

char(*opt=None*)

MeshFem の string description を出力します。

デフォルトでは、 *opt* が 'with_mesh' の場合を除き、リンクされたメッシュオブジェクトの説明は含まれません。

convex_index()

FEM を持つ凸包のリストを返します。

display()

MeshFem オブジェクトの簡単な概要が表示されます。

dof_from_cv(*CVids*)

非推奨の機能です。代わりに MeshFem.basic_dof_from_cv() を使用します。

dof_from_cvid(*CVids=None*)

非推奨の機能です。代わりに MeshFem.basic_dof_from_cvid() を使用します。

dof_from_im(*mim*, *p=None*)

mf と積分法 *mim* で計算される質量行列に大きく寄与する自由度を選択し返します。

p は積分法が作用する次元 (デフォルトは *p* = メッシュの次元) を表します。

重要:levelset と交差しない凸包には、有効な積分法を設定する必要があります!

dof_nodes(*DOFids=None*)

非推奨の機能です。代わりに MeshFem.basic_dof_nodes() を使用します。

dof_on_region(*Rs*)

Rs にリストされたメッシュ領域のいずれかに存在する自由度 (任意削減後) のリストを返

します。

より正確には、この関数は *Rs* (境界領域の場合、一部の節点自由度は正確に境界上に存在しない場合があります。たとえば、 Pk(n,0) の自由度は凸包の中心に存在しますが、基底関数は凸包境界上で null ではありません) に #ids がリストされている領域のいずれかにおいて、サポートが非 NULL である基本的な自由度を返します。

縮退された mesh_fem の場合、自由度は対応する形状関数のポテンシャルがこの領域でゼロでない場合に領域上にあります。拡張行列を使用して、基本自由度と縮退自由度間の対応を作ります。

dof_partition()

'dof_partition' 配列を取得します。

MeshFem の各凸包に整数 (パーティション番号) を関連付ける配列を返します。既定では、配列はすべてゼロです。MeshFem の各凸包の自由度は、同じパーティション番号を持つ隣接する凸包の自由度にのみ接続されるため、部分的に不連続な MeshFem を非常に簡単に作成することができます。

eval(*expression*, *gl={}*, *lo={}*)

（ lagrangian）MeshFem で式を補間します。

例

```
mf.eval('x*y') # interpolates the function 'x*y'
mf.eval('[x,y]') # interpolates the vector field '[x,y]'

import numpy as np
mf.eval('np.sin(x)',globals(),locals()) # interpolates the function␣
↪sin(x)
```

export_to_dx(*filename*, **args*)

概　要: MeshFem.export_to_dx(self,string filename, ...['as', string mesh_name][,'edges']['serie',string serie_name][,'ascii'][,'append'], U, 'name'...)

MeshFem と一部のフィールドを OpenDX ファイルに書き出します。

MeshFem でさまざまな凸包が混在する場合 (つまり、四角形と三角形)、または OpenDX で特定の要素型が処理されない (つまり、プリズム接続は OpenDX では認識されない) 場合、この関数は失敗します。

FEM は次数 1 の Pk(または Qk)FEM にマップされます。高次 FEM または高次幾何変換を表す必要がある場合は、Slice.export_to_dx() を使用する必要があります。

export_to_pos(*filename*, *name=None*, **args*)

概要: MeshFem.export_to_pos(self,string filename[, string name][[,MeshFem mf1], mat U1,

string nameU1[[,MeshFem mf2], mat U2, string nameU2,...]])

MeshFem と一部のフィールドを pos ファイルに書き出します。

FEM および幾何学的変換は、1 次アイソパラメトリック Pk(または Qk)FEM(GMSH はより高次の要素を処理しないため) にマッピングされます。

export_to_vtk (*filename*, **args*)

概要: MeshFem.export_to_vtk(self,string filename, ... ['ascii'], U, 'name'...)

MeshFem と一部のフィールドを vtk ファイルに書き出します。

FEM および幾何変換は、次数 1 または 2 のアイソパラメトリック Pk(または Qk)FEM(VTK では上位の要素を処理できないため) にマッピングされます。高次 FEM または高次幾何変換を表す必要がある場合は Slice.export_to_vtk() を考慮する必要があります。

extend_vector (*V*)

与えられたベクトル V に MeshFem の縮約行列を乗算します。

extension_matrix ()

オプションの拡張行列を返します。

fem (*CVids=None*)

MeshFem で使用される FEM のリストを返します。

FEMs は *CVids* で与えられた凸包にあるすべての Fem オブジェクトの配列です。 *CV2F* が出力引数として指定された場合 *CVids* にリストされている各凸包に対し、 *FEMs* 内の対応する FEM のインデックスが含まれます。

メッシュの一部でない凸包、または FEM を持たない凸包については *CV2F* の対応エントリを -1 に設定します。

has_linked_mesh_levelset ()

mesh_fem_level_set かどうかを確認します。

interpolate_convex_data (*Ucv*)

メッシュの各凸包で指定されたデータを MeshFem の自由度に補間します。MeshFem は lagrangian である必要があり、不連続である必要があります (通常は FEM_PK(N,0) または FEM_QK(N,0) を使用する必要があります)。

入力ベクトル Ucv の最後の次元には Mesh.max_cvid() の要素が必要です。

使用例: MeshFem.interpolate_convex_data(Mesh.quality())

is_equivalent (*CVids=None*)

MeshFem が等価かどうかをテストします。

MeshFem.is_lagrangian() を参照してください。

is_lagrangian(*CVids=None*)

MeshFem が Lagrangian であるかどうかをテストします。

Lagrangian とは、各基底関数 Phi[i] が、 Phi[i](P[j]) = delta(i,j) となることを意味します。ここで P[j] は、j 番目の基底関数の自由度位置であり、i==j なら delta(i,j) = 1 であり、そうでなければ 0 です。

CVids を省略すると、メッシュ内のすべての凸状が Lagrangian の場合に 1 を返します。*CVids* を使用すると、Lagrangian である (*CVids* に関する) 凸包インデックスを返します。

is_polynomial(*CVids=None*)

すべての基本関数が多項式であるかどうかをテストします。

MeshFem.is_lagrangian() を参照してください。

is_reduced()

オプションの縮退マトリックスが自由度に適用される場合は 1 を返します。

linked_mesh()

mf にリンクされたメッシュオブジェクトへの参照を返します。

linked_mesh_levelset()

mesh_fem_level_set の場合はリンクされた mesh_level_set を返します。

memsize()

mesh_fem オブジェクトが使用するメモリ量 (バイト単位) を返します。

この結果では、リンクされたメッシュオブジェクトは考慮されません。

mesh()

mf にリンクされたメッシュオブジェクトへの参照を返します (Mesh.linked_mesh() と同一です)。

nb_basic_dof()

MeshFem の基本自由度 (dof) の数を返します。

nbdof()

MeshFem の自由度 (dof) の数を返します。

non_conformal_basic_dof(*CVids=None*)

部分的にリンクされた自由度を返します。

メッシュの境界上にあるものを除き、1 つの凸包にのみ属する凸包の境界上にある基本自由度を返します。たとえば、凸包の 'a' と 'b' が共通の面を共有し、 'a' に P1 FEM があ

り、'b' に P2 FEM がある場合、面の中央の基本自由度がこの関数によって返されます (これは、古典的 FEM と階層的 FEM の間のインターフェースを探索するときに有用です)。

non_conformal_dof(*CVids=None*)

非推奨の機能です。代わりに MeshFem.non_conformal_basic_dof() を使用してください。

qdim()

MeshFem によって補間されたフィールドの次元 Q を返します。

デフォルトでは、Q=1(スカラー場) です。これは、自由度の番号付けに影響します。

reduce_meshfem(*RM*)

縮退メッシュ数の設定 この関数はマトリックス RM の独立したベクトルの集合を選択することにより、有限要素法の自由度を選択します。RM の列数は、有限要素法の自由度に対応していなければなりません。

reduce_vector(*V*)

提供されたベクトル V に MeshFem の拡張行列を乗算します。

reduction(*s*)

縮退/拡張マトリックスの使用を設定または設定解除します。

reduction_matrices(*R*, *E*)

縮退および拡張マトリックスを設定し、その使用を有効にします。

reduction_matrix()

オプションの縮退行列を返します。

save (*filename*, *opt=None*)

MeshFem をテキストファイル (オプションで *opt* が文字列 'with_mesh' の場合はリンクされたメッシュオブジェクト) に保存します。

set_classical_discontinuous_fem(*k*, **args*)

概要: MeshFem.set_classical_discontinuous_fem(self, int k[[, 'complete'], @tscalar alpha[, ivec CVIDX]])

k 次の古典的 (Lagrange 多項式) 不連続有限要素法を与えます。

FEM_PK_DISCONTINUOUS を使用する場合を除き MeshFem.set_classical_fem() に似ています。パラメータ *alpha* は節点のインセット、 $0 \leq alpha < 1$ で、0 は通常の自由度節点を意味し、値が大きくなると節点は重心に向かって移動し、1 はすべての自由度が重心で破綻することを意味します。 'complete' オプションは、要素の形状変換が不完全なもの (例えば 8 節点四面体または 20 節点六面体) であっても、完全な Langrange 多項式要素が必要です。

set_classical_fem(*k*, **args*)

概要: MeshFem.set_classical_fem(self, int k[[, 'complete'], ivec CVids])

MeshFem に古典的な (Lagrange 多項式) 有限要素法次数 *k* を割り当てます。 'complete' オプションは、要素の形状変換が不完全なもの (例えば 8 節点四辺形または 20 節点六面体) であっても、完全な Langrange 多項式要素が必要です。

simplex に FEM_PK 、parallelepiped に FEM_QK などを使用します。

set_dof_partition(*DOFP*)

'dof_partition' 配列を変更します。

DOFP は MeshFem の各凸包に対する整数値を保持するベクトルです。 "dof partition" については、 MeshFem.dof_partition() を参照してください。

set_enriched_dofs(*DOFs*)

MeshFem 積オブジェクトのみのため。強化された自由度を設定し、MeshFem 積を当てはめます。

set_fem(*f*, *CVids=None*)

有限要素法を設定します。

FEM の *f* を *CVids* に #ids がリストされているすべての凸包に割り当てます。 *CVids* が指定されていない場合、積分はすべての凸包に割り当てられます。

使用可能な FEM メソッドのリストについては、FEM のヘルプを参照してください。

set_partial(*DOFs*, *RCVs=None*)

部分的な MeshFem にのみ適用できます。 *mf* の自由度のサブセットを変更します。

RCVs が与えられた場合、 *RCVs* に列挙されている凸包には FEM は適用されません。

set_qdim(*Q*)

MeshFem によって補間されるフィールドの *Q* 次元を変更します。

Q = 1 は MeshFem がスカラーフィールドを記述することを意味し、 *Q = N* は MeshFem が次数 N のベクトルフィールドを記述することを意味します。

7.11 MeshIm

class MeshIm(**args*)

GeFEM MeshIm オブジェクト

このオブジェクトは、メッシュ全体で定義された積分法を表します (潜在的に境界にあります)。

MeshIm オブジェクトの汎用的なコンストラクタ

- `MIM = MeshIm('load', string fname[, Mesh m])` ファイルから MeshIm をロードします。

  メッシュ *m* が与えられなかった場合 (この種類のファイルにはメッシュは保存されません)、ファイルから読み込まれ、その descriptor が 2 番目の出力引数として返されます。

- `MIM = MeshIm('from string', string s[, Mesh m])` は文字列から MeshIm オブジェクトを作成します。

  `MeshIm.char()` も参照してください。

- `MIM = MeshIm('clone', MeshIm mim)` MeshIm のコピーを作成します。

- `MIM = MeshIm('levelset', MeshLevelSet mls, string where, Integ im[, Integ im_tip[, Integ im_set]])` levelset によって暗黙的に定義されたパーティションに準拠する積分法を構築します。

  where 引数は levelset に関する積分領域を定義します。この引数は 'ALL', 'INSIDE', 'OUTSIDE' および 'BOUNDARY' の中から選択する必要があります。

  複数の levelset がある場合は、ブール演算を定義する文字列を使用して積分領域を定義できます。

  構文は非常に単純です。たとえば、3 つの異なるレベルセットがある場合、

  "a*b*c" は各 levelset によって定義された領域の交差です (この関数が呼び出されない場合のデフォルトの動作です)。

  "a+b+c" は領域の結合です。

  "c-(a+b)" は第 3levelset の領域から他の二つの領域の和集合を引いたものです。

  "!a" は (すなわち a(x)>0 である) 領域の補完領域です。

  最初の levelset は常に "a" で参照され、次のレベルセットは "b" で参照されます。

  インスタンス INSIDE(a*b*c) 用

  注意: この積分法は level-set で切り取られた要素にのみ定義されます。'ALL'、'INSIDE' と 'OUTSIDE' オプションの場合は、必ず `MeshIm.set_integ()` メソッドを使って残りの要素の積分法を定義する必要があります。

- `MIM = MeshIm(Mesh m, [{Integ im|int im_degree}])` 新しい MeshIm オブジェクトを作成します。

  便宜上、オプションの引数 (*im* または *im_degree*) を指定することもできます。`MeshIm.integ()` への呼び出しがこれらの引数とともに出力されます。

adapt()
MeshIm levelset オブジェクトの場合のみ使用します。levelset 関数の変更に合わせて積分法を変更します。

char()
MeshIm の説明を文字列で出力します。

既定値では、リンクされたメッシュオブジェクトの説明は含まれません。

convex_index()
積分法がある凸包のリストを返します。

ダミーの IM_NONE メソッドを持つ凸包は表示されません。

display()
MeshIm オブジェクトの簡単な概要が表示されます。

eltm(*em*, *cv*, *f=None*)
凸包の *cv* 上で積分された基本行列 (またはテンソル) を返します。

警告

Be sure that the fem used for the construction of *em* is compatible with the fem assigned to element *cv* ! This is not checked by the function ! If the argument *f* is given, then the elementary tensor is integrated on the face *f* of *cv* instead of the whole convex.

im_nodes(*CVids=None*)
積分点の座標とその重みを返します。

CVids は Mesh.region() によって返される凸包のリストまたは凸包面のリストです。

警告

メッシュの一部でない凸包、または近似積分法を持たない凸包には、対応するエントリ (これは、厳密な積分法には意味がありません!) がありません。

integ(*CVids=None*)
MeshIm で使用される積分法のリストを返します。

I は *CVids* で与えられた凸包にあるすべての整数オブジェクトの配列です。 *CV2I* が出力引数として与えられた場合、 *CVids* に列挙されている各凸包に対し、 *I* における相関積分法のインデックスを含みます。

メッシュの一部ではない凸包、または積分法を持たない凸包については、 *CV2I* の対応エントリを-1 に設定します。

linked_mesh()
mim にリンクされている Mesh オブジェクトへの参照を返します。

memsize()
MeshIm オブジェクトによって使用されるメモリの量 (バイト単位) を返します。

この結果では、リンクされたメッシュオブジェクトは考慮されません。

save(*filename*)
MeshIm をテキストファイル (とオプションでリンクされたメッシュオブジェクト) に保存します。

set_integ(**args*)
概要: MeshIm.set_integ(self,{Integ imlint im_degree}[, ivec CVids])

積分法を設定します。

CVids に#id がリストされているすべての凸包に積分法を割り当てます。 *CVids* が指定されていない場合、積分はすべての凸包に割り当てられます。 Integ('IM_SOMETHING') から取得した積分法ハンドル *im* を用いて、特定の積分法を割り当てることもできるし、 *im_degree* を用いて、getfem に適切な積分法を選択させることもできます。(degree $\leq$ im_degree の多項式が正確に積分されるように選択します。 *im_degree=-1* の場合、ダミーの積分法 IM_NONE が使用されます。)。

7.12 MeshImData

class MeshImData(**args*)
GeFEM MeshImData オブジェクト

このオブジェクトは mesh_im オブジェクトで定義されたデータを表します。

MeshImData オブジェクトの汎用的なコンストラクタ

- `MIMD = MeshImData(MeshIm mim, int region, ivec size)` MeshIm オブジェクトにリンクされた新しい MeshImd オブジェクトを構築します。 *region* が与えられた場合、考慮された積分点はこの領域でフィルタリングされます。 *size* は整数のベクトルであり、積分点ごとに格納されるデータの次元を指定します。指定しない場合、スカラー格納データが考慮されます。

display()
MeshImd オブジェクトの簡単な概要が表示されます。

linked_mesh()
mim にリンクされている Mesh オブジェクトへの参照を返します。

nb_tensor_elements()
保存されているデータのサイズを出力します (各積分点)。

nbpts()
積分点の数を出力します (考慮された領域でフィルタリングされます)。

region()
MeshImd が制約されている領域を出力します。

set_region(*rnum*)
考慮する領域を *rnum* に設定します。

set_tensor_size()
積分点ごとのデータのサイズを設定します。

tensor_size()
保管されたデータの次元を出力します (積分点当たり)。

7.13 MeshLevelSet

class MeshLevelSet(**args*)
GeFEM MeshLevelSet オブジェクト

mesh_levelset オブジェクトの汎用的なコンストラクタです。このオブジェクトの役割は、一定数の level_set によるメッシュカットを提供することです。このオブジェクトは等角積分法 (オブジェクト mim および enrich 有限要素法 (XFEM)) の構築に使用されます。

MeshLevelSet オブジェクトの汎用的なコンストラクタ

- `MLS = MeshLevelSet(Mesh m)` メッシュから新しい MeshLevelSet オブジェクトを作成し、そのハンドルを返します。

adapt()
すべての作業を行います (levelset で凸包を切り取る)。

MeshLevelSet オブジェクトを初期化したり、任意の levelset 関数の値が変更されたときにそれを実現するには、このメソッドを呼び出す必要があります。

add(*ls*)
LevelSet *ls* へのリンクを追加します。

参照のみが保持され、コピーは行われません。リンクされたメッシュが LevelSet によってカットされていることを示すには、 *ls* が LevelSet オブジェクトであるこのメソッドを呼び出す必要があります。任意の数の LevelSet を追加できます。

警告

ls のメッシュとリンクされたメッシュは同じでなければなりません。

char()
MeshLevelSet の (ユニークな) 文字列表現を 出力します。

これを 使用して、2 つの異なる MeshLevelSet オブジェクト 間の比較を 実行できます。この機能は完成予定です。

crack_tip_convexes()
リンクされたメッシュの凸包のリンクされた LevelSet の先端を 持つ #id のリストを 返します。

cut_mesh()
リンクされた LevelSet でカットされたメッシュを返します。

display()
MeshLevelSet オブジェクト の簡単な概要が表示されます。

levelsets()
リンクされた LevelSet への参照のリストを 返します。

linked_mesh()
リンクされたメッシュへの参照を返します。

memsize()
MeshLevelSet で使用されているメモリ量 (バイト 単位) を返します。

nb_ls()
リンクされた LevelSet の数を返します。

sup(*ls*)
レベルセット *ls* へのリンクを削除します。

7.14 MesherObject

class MesherObject(**args*)
GeFEM MesherObject オブジェクト

このオブジェクト は、Getfem の実験的メッシュ手法でメッシングされるジオメトリックオブジェクトを 表します。

MesherObject オブジェクト 用の汎用的なコンストラクタ

- `MF = MesherObject('ball', vec center, scalar radius)` は中心と 半径に対応する球を 表します。

- `MF = MesherObject('half space', vec origin, vec normal_vector)` は原点を含み、 *normal_vector* に垂直な面で区切られた半空間を表します。選択したパーツは、法線ベクトル方向です。これにより、ポリゴンや多面体を構築するなど、平面を使用してジオメトリを切り取ることができます。
- `MF = MesherObject('cylinder', vec origin, vec n, scalar length, scalar radius)` 原点、ベクトル *n* 、長さによって軸が決まる特定の半径の円柱 (どの次元でも) を表します。
- `MF = MesherObject('cone', vec origin, vec n, scalar length, scalar half_angle)` 原点、ベクトル *n* 、長さによって軸が決まる特定の (ラジアン単位の) 半角の (各次元の) 円錐を表します。
- `MF = MesherObject('torus', scalar R, scalar r)` は Z 軸に沿った 3 次元軸のトーラスを表します。大きな半径 'R' と小さな半径 *r* を持ちます。現時点では、軸を変更することはできません。
- `MF = MesherObject('rectangle', vec rmin, vec rmax)` 軸に平行な長方形 (または 3 次元の平行六面体) を表します。
- `MF = MesherObject('intersect', MesherObject object1 , MesherObject object2, ...)` 複数のオブジェクトの交差。
- `MF = MesherObject('union', MesherObject object1 , MesherObject object2, ...)` 複数のオブジェクトの結合。
- `MF = MesherObject('set minus', MesherObject object1 , MesherObject object2)` object1 と object2 の差のジオメトリックオブジェクト。

char()

MesherObject の (ユニークな)string representation を出力します。

これを使用して、2 つの異なる MesherObject オブジェクト間の比較を実行できます。この機能は完成予定です。

display()

MesherObject オブジェクトの概要が表示されます。

7.15 Model

class Model(**args*)

GeFEM Model オブジェクト

モデル変数には、変数、状態データ、モデルの説明が格納されます。これには、全体接線行列、RHS、および制約条件が含まれます。モデルには *real* モデルと *complex* モデルの 2 つの種類があります。

Model オブジェクトの汎用的なコンストラクタ

- `MD = Model('real')` 実数が未知数のモデルを作ります。
- `MD = Model('complex')` 複素数が未知数のモデルを構築します。

Neumann_term(*varname*, *region*)

region の fem 変数 *varname* の Neumann 項に対応するアセンブリ文字列を返します。モデルブリックによって宣言されたアセンブリ文字列から推定されます。 *region* は *varname* が定義されているメッシュ上の境界領域のインデックスである必要があります。この関数は、すべての体積ブリックが宣言された後で呼び出すようにしてください。ブリックがアセンブリ文字列の宣言を省略した場合はエラーが発生します。

add_Dirichlet_condition_with_Nitsche_method(*mim*, *varname*, *Neumannterm*, *datagamma0*, *region*, *theta=None*, **args*)

概要: ind = Model.add_Dirichlet_condition_with_Nitsche_method(self, MeshIm mim, string varname, string Neumannterm, string datagamma0, int region[, scalar theta][, string dataname])

変数 *varname* とメッシュ領域 *region* に Dirichlet 条件を追加します。この領域は境界である必要があります。Neumann 項は、高水準汎用アセンブリ言語の表現である (Green 公式により得られる)Neumann 項の式です。すべての体積ブリックをモデルに追加すると、この項は Model.Neumann_term(varname, region) によって得られます 。Dirichlet 条件は Nitsche 法により処理されます。 *datag* は Dirichlet 条件のオプションの RHS です。 *datagamma0* は Nitsche 法のパラメータです。 *theta* は正または負のスカラー値です。 *theta = 1* は *gamma0* が小さい場合に条件的が強制される標準的な対称法です。 *theta = -1* は無条件に強制的な skew 対称法に対応します。 *theta = 0* (デフォルト) は非線形問題に対しても Neumann 項の二次導関数を必要としない最も単純な方法です。モデル内のブリックインデックスを返します。

add_Dirichlet_condition_with_multipliers(*mim*, *varname*, *mult_description*, *region*, *dataname=None*)

変数 *varname* とメッシュ領域 *region* に Dirichlet 条件を追加します。この領域は境界である必要があります。Dirichlet 条件は *mult_description* によって記述される乗数変数で規定されます。 *mult_description* が文字列の場合は、乗数 (これは、最初にモデルのメッシュ領域で乗数変数として宣言する必要があります) に対応する変数名と見なされます。有限要素法 (mesh_fem オブジェクト) の場合は、乗数変数がモデルに追加され、この有限要素法 (メッシュ領域領域で制限され、最終的に他の乗数変数との自由度の競合が抑制されま

す) に基づいて構築されます。整数の場合は、乗数変数がモデルに追加され、その整数の次数の従来の有限要素に基づいて構築されます。 *dataname* は Dirichlet 条件のオプションの RHS です。定数の場合もあれば、fem で記述される場合もあります。スカラー値またはベクトル値で、Dirichlet 条件が指定されている変数によって異なります。モデル内のブリックインデックスを返します。

add_Dirichlet_condition_with_penalization(*mim*, *varname*, *coeff*, *region*, *dataname=None*, *mf_mult=None*)

変数 *varname* とメッシュ領域 *region* に Dirichlet 条件を追加します。この領域は境界である必要があります。Dirichlet 条件はペナルティとともに処理されます。ペナルティ係数は初期値は *coeff* であり、モデルのデータに追加されます。*dataname* は Dirichlet 条件のオプションの右辺です。定数の場合もあれば、有限要素法で記述される場合もあります。スカラー値またはベクトル値で、Dirichlet 条件が指定されている変数によって異なります。*mf_mult* はオプションのパラメータで、乗数空間を指定して Dirichlet 条件を弱めることができます。モデル内のブリックインデックスを返します。

add_Dirichlet_condition_with_simplification(*varname*, *region*, *dataname=None*)

変数 *varname* とメッシュ領域 *region* に (単純な)Dirichlet 条件を追加します。Dirichlet 条件は、(0 は対角線の外側、1 はマトリックスの対角線、期待値は右側) 上の変数の自由度に対応する行を修正することからなる最終的な線形系 (非線形問題の接線系) の簡単な後処理によって規定されます。線形システムの対称性は、他のすべてのブリックが対称である場合に維持されます。このブリックは、単純な Dirichlet 条件 (一致境界で宣言された自由度のみが指定されています) 用に予約されています。このブリックの還元された自由度への適用は問題があるかもしれません。固有ベクトル有限要素法はサポートされていません。 *dataname* はディリクレ条件のオプションの右側です。これは、定数 (この場合は、Lagrange fem にのみ適用できます) または *varname* と同じ有限要素法で記述された (重要な) にすることができます。モデル内のブリックのインデックスを返します。

add_Fourier_Robin_brick(*mim*, *varname*, *dataexpr*, *region*)

変数 *varname* を基準にして、モデルに Fourier-Robin 項を追加します。これは以下の形式の弱項に相当します $\int (qu).v$ 。*dataexpr* はパラメータです。高水準汎用アセンブリ言語 (ただし、モデルのデータである必要がある複素数バージョンは除きます) の任意の有効な表現を使用できます。 *region* は項が追加されるメッシュ領域です。モデル内のブリックインデックスを返します。

add_HHO_reconstructed_gradient(*transname*)

このモデルに HHO 法に対する勾配の再構成に対応する基本変換を加えました。名前は基本変換に付けられた名前です。

add_HHO_reconstructed_symmetrized_gradient(*transname*)

このモデルに HHO 法に対する勾配の再構成に対応する基本変換を加えました。名前は基

本変換に付けられた名前です。

add_HHO_reconstructed_symmetrized_value(*transname*)
対称化勾配を用いた HHO 法に対する変数の再構成に対応する素変換をモデルに加えます。名前は基本変換に付けられた名前です。

add_HHO_reconstructed_value(*transname*)
このモデルに HHO 法に対する変数の再構成に対応する基本変換を加えました。名前は基本変換に付けられた名前です。

add_HHO_stabilization(*transname*)
このモデルに HHO 安定化演算子に対応する素変換を加えます。名前は、基本変換に付けられた名前です。

add_HHO_symmetrized_stabilization(*transname*)
対称化勾配を用いた HHO 安定化演算子に対応する素変換をモデルに加えます。名前は、基本変換に付けられた名前です。

add_Helmholtz_brick(*mim*, *varname*, *dataexpr*, *region=None*)
変数 *varname* に対して相対的に Helmholtz 項をモデルに追加します。 *dataexpr* は波数です。 *region* はオプションの項が追加されるメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。モデル内のブリックインデックスを返します。

add_Houbolt_scheme(*varname*)
変数 ‘varname‘の時間離散化に Houbolt 法を適用します。変数の 2 次時間導関数が多くても存在する場合にのみ有効です。

add_Kirchhoff_Love_Neumann_term_brick(*mim*, *varname*, *dataname_M*, *dataname_divM*, *region*)
変数 *varname* とメッシュ領域 *region* に Kirchhoff-Love モデルの Neumann 項ブリックを追加します。 *dataname_M* は曲げモーメントテンソルを表し、 *dataname_divM* はその発散を表します。モデル内のブリックインデックスを返します。

add_Kirchhoff_Love_plate_brick(*mim*, *varname*, *dataname_D*, *dataname_nu*, *region=None*)
変数 *varname* とメッシュ領域 *region* にバイラプラシアンブリックを追加します。これは、項 $\Delta(D\Delta u)$ を表し、ここで $D(x)$ は *dataname_D* によって決定される曲げ弾性率です。この項は、 *dataname_nu* をポアソン比とする Kirchhoff-Love プレートモデルに従って部分的に積分されています。モデル内のブリックインデックスを返します。

add_Laplacian_brick(*mim*, *varname*, *region=None*)
変数 *varname* に相対的に Laplacian 項をモデルに追加します (実際には負 $-\text{div}(\nabla u)$ です)。これがベクトル値の変数である場合、Laplace 項がコンポーネントごとに追加されます。 *region* は項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。モデル内のブリックインデックスを返します。

add_Mindlin_Reissner_plate_brick(*mim*, *mim_reduced*, *varname_u3*, *varname_theta*, *param_E*, *param_nu*, *param_epsilon*, *param_kappa*, *variant=None*, **args*)

概要: ind = Model.add_Mindlin_Reissner_plate_brick(self, MeshIm mim, MeshIm mim_reduced, string varname_u3, string varname_theta , string param_E, string param_nu, string param_epsilon, string param_kappa [,int variant [, int region]])

古典的な Reissner-Mindlin プレート モデルに対応する項を追加します。ここで、 *varname_u3* は横方向の変位、*varname_theta* は中立面に垂直なファイバーの回転、'param_E' はヤング率、 *param_nu* はポアソン比、 *param_epsilon* はプレートの厚さ、 *param_kappa* はせん断補正係数です。このブリックは高水準汎用アセンブリ言語を使用しているため、パラメータをこの言語の正規表現にすることができます。3 つのバリエーションがあります。 *variant = 0* は非簡約化された定式化に対応し、その場合には積分法 *mim* のみが使用されます。実際には、この変形は強いロック現象の影響を受けるため、使用できません。 *variant = 1* は、 *mim* が回転項に使用され、 *mim_reduced* が横せん断項に使用される縮小積分に対応します。 *variant = 2* (デフォルト) は、横せん断項の回転 RT0 要素への投影に対応します。現時点では、(三角形要素のロック現象を除去するだけでは不十分であるため) 四角形のみに適用されます。また、高次要素を使用する場合、RT0 への投影によって近似の次数が減少することにも注意してください。モデル内のブリックのインデックスを返します。

add_Newmark_scheme(*varname*, *beta*, *gamma*)

変数 'varname'の時間離散化に theta 法を適用します。変数の 2 次時間導関数が多くても存在する場合にのみ有効です。

add_Nitsche_contact_with_rigid_obstacle_brick(*mim*, *varname*, *Neumannterm*, *dataname_obstacle*, *gamma0name*, *region*, *theta=None*, **args*)

概要: ind = Model.add_Nitsche_contact_with_rigid_obstacle_brick(self, MeshIm mim, string varname, string Neumannterm, string dataname_obstacle, string gamma0name, int region[, scalar theta[, string dataname_friction_coeff[, string dataname_alpha, string dataname_wt]]])

Coulomb 摩擦の有無にかかわらず、接触条件を変数 *varname* とメッシュ境界 *region* に追加します。接触条件は Nitsche 法で規定されています。剛体障害物は、データ *dataname_obstacle* が障害物までの符号付き距離 (有限要素法による補間) で記述されるべきである。 *gamma0name* は Nitsche 法のパラメータです。 *theta* は正または負のスカラー値です。 *theta = 1* は *gamma0* が小さい場合に条件的に強制される標準的な対称法に相当する。*theta=-1* は無条件に強制的なスキュー対称法に対応する。 *theta = 0* は

Neumann 項の 2 次導関数を必要としない最も単純な方法です。オプションのパラメータ *dataname_friction_coeff* は摩擦係数で、一定にすることも、有限要素法で定義することもできます。注意: このブリックは Neumann 項を持つ偏微分項に対応するすべてのブリックの後にモデルに追加する必要があります。さらに、このブリックは Neumann 項を宣言するブリックにのみ適用できます。モデル内のブリックのインデックスを返します。

add_Nitsche_fictitious_domain_contact_brick(*mim*, *varname1*, *varname2*, *dataname_d1*, *dataname_d2*, *gamma0name*, *theta=None*, **args*)

概要: ind = Model.add_Nitsche_fictitious_domain_contact_brick(self, MeshIm mim, string varname1, string varname2, string dataname_d1, string dataname_d2, string gamma0name [, scalar theta[, string dataname_friction_coeff[, string dataname_alpha, string dataname_wt1,string dataname_wt2]]])

仮想ドメイン内の 2 つのボディ間にクーロン摩擦の有無による接触条件を追加します。接触条件は、Nitsche 法では変数 *varname_u1* が第 1 およびスレーブボディに対応し、Nitsche 法では変数 *varname_u2* が第 2 およびマスターボディに対応します。接触条件を仮想スレーブ境界上で評価しました。最初のボディはレベルセット *dataname_d1* によって記述され、2 番目のボディはレベルセット *dataname_d2* によって記述されます。 *gamma0name* は Nitsche 法のパラメータです。 *theta* は正または負のスカラー値です。 *theta = 1* は *gamma0* が小さい場合に条件的に強制される標準的な対称法に相当する。 *theta = -1* は無条件に強制的なスキュー対称法に対応する。 *theta = 0* は Neumann 項の 2 次導関数を必要としない最も単純な方法です。オプションのパラメータ *dataname_friction_coeff* は摩擦係数で、一定にすることも、有限要素法で定義することもできます。注意: このブリックは Neumann 項を持つ偏微分項に対応するすべてのブリックの後にモデルに追加する必要があります。さらに、このブリックは Neumann 項を宣言するブリックにのみ適用できます。モデル内のブリックのインデックスを返します。

add_Nitsche_large_sliding_contact_brick_raytracing(*unbiased_version*, *dataname_r*, *release_distance*, *dataname_fr=None*, **args*)

概要: ind = Model.add_Nitsche_large_sliding_contact_brick_raytracing(self, bool unbiased_version, string dataname_r, scalar release_distance[, string dataname_fr[, string dataname_alpha[, int version]]])

Nitsche 法に基づいて、摩擦ブリックと有限滑り接触をモデルに追加します。このブリックは自己接触、複数の変形可能体間の接触及び剛体障害物との接触に対処できます。上位

レベルの汎用アセンブリを使用します。モデルに raytracing_interpolate_transformation オブジェクトを追加します。 "unbiased_version" とは、使用する Nitsche 法のバージョンのことです。(バイアスされていない、またはバイアスされているもの)。スレーブ境界ごとに、この境界上の表示変数の関数として材料法則を定義する必要があります。放出距離は慎重に決定すべきです (一般に平均要素サイズの数倍で、ボディの厚さよりも薄いです)。最初は、ブリックは接触境界なしで追加されます。接触境界と剛体には、特別な機能が追加されています。 *version* は非対称バージョンでは 0(デフォルト値)、より対称なバージョンでは 1(摩擦がなくても完全に対称的でない) です。

add_Nitsche_midpoint_contact_with_rigid_obstacle_brick(*mim*, *varname*, *Neuman-nterm*, *Neuman-nterm_wt*, *dataname_obstacle*, *gamma0name*, *region*, *theta*, *dataname_friction_coeff*, *dataname_alpha*, *dataname_wt*)

実験的な BRICK: 中間点法専用!! Coulomb 摩擦の有無にかかわらず、接触条件を変数 *varname* とメッシュ境界 *region* に追加します。接触条件は Nitsche 法で規定されています。剛体障害物は、データ *dataname_obstacle* が障害物までの符号付き距離 (有限要素法による補間) で記述されるべきです。 *gamma0name* は Nitsche 法のメソッドパラメータです。 *theta* は正または負のスカラー値です。 *theta = 1* は *gamma0* が小さい場合に条件的に強制される標準的な対称法に相当します。 *theta = -1* は無条件に強制的なスキュー対称法に対応する。 *theta = 0* は Neumann 項の 2 次導関数を必要としない最も単純な方法です。オプションのパラメータ *dataname_friction_coeff* は摩擦係数で、一定にすることも、有限要素法で定義することもできます。モデル内のブリックのインデックスを返します。

add_assembly_assignment(*dataname*, *expression*, *region=None*, **args*)

概要: Model.add_assembly_assignment(self, string dataname, string expression[, int region[, int order[, int before]]])

アセンブリ時に評価され、im_data 型であるデータ *dataname* に割り当てられる式 *expr* を追加します。これにより、インスタンスは、他のアセンブリで使用するアセンブリ計算のサブ式を保存できます。たとえば、塑性モデルに塑性歪みを保存するために使用できます。order は、この割り当てを行うアセンブリの次数 (ポテンシャル (0)、弱形式 (1)、接線システム (2)、または各次数 (-1)) を表します。デフォルト値は 1 です。before=1 の場合、

他のアセンブリ項の計算の前に割り当てが実行され、データが残りのアセンブリで中間結果として使用できるようになります (接線システム式の微分が実行されず、データとみなされることに注意してください)。before=0(デフォルト) の場合、代入はアセンブリ項の後に行われます。

add_basic_contact_brick(*varname_u*, *multname_n*, *multname_t=None*, **args*)

概要: ind = Model.add_basic_contact_brick(self, string varname_u, string multname_n[, string multname_t], string dataname_r, Spmat BN[, Spmat BT, string dataname_friction_coeff][, string dataname_gap[, string dataname_alpha[, int augmented_version[, string dataname_gamma, string dataname_wt]]])

摩擦ブリック付きまたは摩擦ブリックなしの接触をモデルに追加します。U が、片側拘束が適用される自由度のベクトルである場合、行列 *BN* はこの拘束が $B_N U \leq 0$ によって定義されるようでなければなりません。摩擦条件は、3 つのパラメータ *multname_t* 、 *BT* 、 *dataname_friction_coeff* を追加することで考慮できます。この場合、接線変位は $B_T U$ であり、行列 *BT* は、 *BN* に $d-1$ を掛けた数だけの行を持たなければなります。ここで、d はドメインの次元です。ここでも、 *dataname_friction_coeff* は摩擦係数を表すデータです。各接触条件の値を表すスカラーまたはベクトルです。一方的制約は、その次元が *BN* の行数に等しくなければならない乗算器 *multname_n* のおかげで規定される。摩擦条件が追加される場合には、その次元が *BT* の行数に等しくなければならない乗算器 *multname_t* によって規定されます。拡張パラメータ *r* は許容値 (Getfem ユーザマニュアルを参照) の範囲内で選択するべきです。 *dataname_gap* は初期ギャップを表すオプションのパラメータです。単一の値または値のベクトルを指定できます。 *dataname_alpha* は増大パラメータ (Getfem ユーザマニュアルを参照) のためのオプションの均質化パラメータである。パラメータ *augmented_version* は拡大戦略を示します。1 は非対称 Alart-Curnier 拡大ラグランジアン、2 は対称的なもの (接触と Coulomb 摩擦の結合を除きます)、3 は拡大乗算器を用いた非対称方法、4 は拡大乗算器と De Saxce 射影を用いた非対称方法です。

add_basic_contact_brick_two_deformable_bodies(*varname_u1*, *varname_u2*, *multname_n*, *dataname_r*, *BN1*, *BN2*, *dataname_gap=None*, **args*)

概要: ind = Model.add_basic_contact_brick_two_deformable_bodies(self, string varname_u1, string varname_u2, string multname_n, string dataname_r, Spmat BN1, Spmat BN2[, string dataname_gap[, string dataname_alpha[, int augmented_version]]])

2 つの変形可能モデル間に摩擦なし接触条件を追加します　体。U1、U2 が片側拘束が適用される自由度のベクトルである場合、行列 *BN1* および *BN2* は、この条件が $B_{N1} U_1 B_{N2} U_2 + le gap$ によって定義されるようでなければならない。この制約は、次元が *BN* の行数に等しくなければならない乗数 *multname_n* によって規定され

ます。拡張パラメータ *r* は許容値 (Getfem ユーザマニュアルを参照) の範囲内で選択されるべきです。 *dataname_gap* は初期ギャップを表すオプションのパラメータです。単一の値または値のベクトルを指定できます。 *dataname_alpha* は、増大パラメータ (Getfem ユーザマニュアルを参照) のためのオプションの均質化パラメータである。パラメータ *aug_version* は拡大法を示します。1 は非対称 Alart-Curnier 拡大 Lagrangian、2 は対称 Lagrangian、3 は拡大乗数を伴う非対称法です。

add_bilaplacian_brick(*mim*, *varname*, *dataname*, *region=None*)

変数 *varname* とメッシュ領域 *region* に bilaplacian ブリックを追加します。これは、$\Delta(D\Delta u)$ という項を表しています。ここで、 $D(x)$ は *dataname* によって決定される係数であり、定数であるか、有限要素法で記述されます。対応する弱形式は $\int D(x)\Delta u(x)\Delta v(x)dx$ の通りです。モデル内のブリックインデックスを返します。

add_constraint_with_multipliers(*varname*, *multname*, *B*, **args*)

概要: ind = Model.add_constraint_with_multipliers(self, string varname, string multname, Spmat B, {vec L | string dataname})

以前にモデルに追加した乗数 *multname* を使い、変数 *varname* (は固定サイズ変数でなければなりません) に陽な制約を追加します。制約は $BU = L$ です。ただし、 *B* は矩形の疎行列です。 Model.set_private_matrix() と Model.set_private_rhs() メソッドを使用すると、いつでも拘束を変更できます。 *L* の代わりに *dataname* が指定された場合、ベクトル *L* は指定された名前のデータとしてモデル内で定義されます。モデル内のブリックインデックスを返します。

add_constraint_with_penalization(*varname*, *coeff*, *B*, **args*)

概要: ind = Model.add_constraint_with_penalization(self, string varname, scalar coeff, Spmat B, {vec L | string dataname})

変数 *varname* に追加の明示的ペナルティ制約を追加します。制約は以下の通りである。 :math‘BU=L‘ と *B* は矩形の疎行列である。 *B* にはペイン行を含めないでください。そうしないと、接線行列全体が単純になります。 Model.set_private_matrix() と Model.set_private_rhs() メソッドを使用すると、いつでも拘束を変更できます。 Model.change_penalization_coeff() を使用できます。 *L* の代わりに *dataname* が指定された場合、ベクトル *L* は指定された名前のデータとしてモデル内で定義されます。モデル内のブリックインデックスを返します。

add_contact_boundary_to_unbiased_Nitsche_large_sliding_contact_brick(*indbrick*, *mim*, *region*, *dispname*, *lambdaname*, *wname=N*

接触境界をマスターとスレーブの両方である摩擦ブリックを持つ既存のバイアスされていない Nitschelarge スライド接触に追加します。

add_contact_with_rigid_obstacle_brick(*mim*, *varname_u*, *multname_n*, *multname_t=None*, **args*)

概要: ind = Model.add_contact_with_rigid_obstacle_brick(self, MeshIm mim, string varname_u, string multname_n[, string multname_t], string dataname_r[, string dataname_friction_coeff], int region, string obstacle[, int augmented_version])

非推奨機能です。代わりに ' 硬質障害物ブリックに節点接触を追加するには' を使用してください。

add_data(*name*, *size*)

固定サイズのデータをモデルに追加します。 *sizes* は整数 (スカラーまたはベクトルデータ用) かテンソルデータの次元のベクトルです。 *name* はデータ名です。

add_elastoplasticity_brick(*mim*, *projname*, *varname*, *previous_dep_name*, *datalambda*, *datamu*, *datathreshold*, *datasigma*, *region=None*)

上位レベルの汎用アセンブリを使用しない古い (廃止予定の) ブリック。等方性材料および準静的モデルの場合、微小変形で変数 *varname* に対して相対的に非線形弾塑性項をモデルに追加します。 *projname* は使用される投影のタイプです。 'VM' または 'Von Mises' で使用できるのは Von Mises 投影だけです。 *datasigma* はマテリアルの制約を表す変数です。 *previous_dep_name* は、前のタイムステップでのディスプレイスメントを表します。さらに、 *varname* が記述される有限要素法は、K 次の mesh_fem であり、 *datasigma* は、少なくとも K-1 次の mesh_fem でなければなりません。 *datalambda* および *datamu* は検討した材料の Lame 係数である。 *datathreshold* は材料の塑性閾値です。最後の 3 つの変数は、定数であったり、同じ有限要素法で記述されていたりします。 *region* は項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。モデル内のブリックインデックスを返します。

add_element_extrapolation_transformation(*transname*, *source_mesh*, *elt_corr*)

恒等変換を表すが、多項式外挿によって現在の要素以外の別の要素で式を評価できる特別

な補間変換を追加します。仮想領域応用における安定化項に使用されます。配列 elt_cor は 2 つのエントリからなる配列で、最初の行は変換に関係する要素を含み、2 番目の行は外挿が必要なそれぞれの要素を含みます。要素が elt_cor にリストされていない場合、評価は現在の要素に対してのみ行われます。

add_elementary_P0_projection(*transname*)

投影 P0 要素に対応する基本変換を追加します。名前は、基本変換に付けられた名前です。

add_elementary_rotated_RT0_projection(*transname*)

2 次元エレメントの回転 RT0 エレメント上の投影に対応する基本変換をモデルに追加します。名前は、基本変換に付けられた名前です。

add_enriched_Mindlin_Reissner_plate_brick(*mim*, *mim_reduced1*, *mim_reduced2*, *varname_ua*, *varname_theta*, *varname_u3*, *varname_theta3*, *param_E*, *param_nu*, *param_epsilon*, *variant=None*, **args*)

概要: ind = Model.add_enriched_Mindlin_Reissner_plate_brick(self, MeshIm mim, MeshIm mim_reduced1, MeshIm mim_reduced2, string varname_ua, string varname_theta,string varname_u3, string varname_theta3 , string param_E, string param_nu, string param_epsilon [,int variant [, int region]])

enriched Reissner-Mindlin プレートモデルに対応する項を追加します。ここで、 *varname_ua* は膜変位、 *varname_u3* は横変位、 *varname_theta* は中間平面に垂直なファイバーの回転、'param_E' は Young 率、*param_nu* は Poisson 比、 *param_epsilon* は板厚です。このブリックは高水準汎用アセンブリ言語を使用しているため、パラメータをこの言語の正規表現にすることができます。4 つのバリエーションがあります。 *variant = 0* は非簡約化された定式化に対応し、その場合には積分法 *mim* のみが使用されます。実際には、この変形は強いロック現象の影響を受けるため、使用できません。 *variant = 1* は、*mim* が回転項に使用され、 *mim_reduced1* が横せん断項に使用され、 *mim_reduced2* がピンチ項に使用される縮小積分に対応します。 *variant = 2* (デフォルト) は、横せん断項の回転 RT0 要素上への投影と、ピンチ項の縮小積分に対応します。現時点では、(三角形要素のロック現象を除去するだけでは不十分であるため) 四角形のみに適用されます。また、高次要素を使用する場合、RT0 への投影によって近似の次数が減少することにも注意してください。 *variant = 3* は、横せん断項の回転 RT0 要素への投影と、ピンチ項の P0 要素への投影とに対応します。現時点では、(三角形要素のロック現象を除去するだけでは不十分であるため) 四角形のみに適用されます。また、高次要素を使用する場合、RT0 への投影によって近似の次数が減少することにも注意してください。モデル内のブリックのインデックスを返します。

add_explicit_matrix(*varname1*, *varname2*, *B*, *issymmetric=None*, **args*)

概要: ind = Model.add_explicit_matrix(self, string varname1, string varname2, Spmat B[, int issymmetric[, int iscoercive]])

変数 *varname1* と *varname2* に対して相対的に接線線形システムに追加される陽な行列を表すブリックを追加します。与えられた行列は *varname1* の次元と同じ数の行と *varname2* の次元と同じ数の列を持つ必要があります。もし 2 つの変数が異なっていて、かつ *issymmetric* が 1 に設定されているならば、行列の転置もまた接線系に加えられます (デフォルトは 0 です)。項が接線系の保磁力に影響を与えない場合 (デフォルトは 0 です)、 *iscoercive* を 1 に設定します。マトリックスは、 Model.set_private_matrix() コマンドで変更できます。モデル内のブリックインデックスを返します。

add_explicit_rhs(*varname*, *L*)

変数 *varname* に対して相対的に接線線形システムの右側に追加される明示的な右辺を表すブリックを追加します。与えられた右辺は *varname* の次元と同じ大きさでなければなりません。右辺は Model.set_private_rhs() コマンドで変更できます。 *L* の代わりに *dataname* が指定された場合、ベクトル *L* は指定された名前のデータとしてモデル内で定義されます。モデル内のブリックインデックスを返します。

add_fem_data(*name*, *mf*, *sizes=None*)

MeshFem にリンクされたモデルにデータを追加します。 *name* はデータ名であり、 *sizes* はオプションのパラメータであり、 *mf* に関して補間的な次元の整数またはベクトルです。

add_fem_variable(*name*, *mf*)

MeshFem にリンクされたモデルに変数を追加します。 *name* は変数名です。

add_filtered_fem_variable(*name*, *mf*, *region*)

MeshFem にリンクされたモデルに変数を追加します。変数は、領域上の自由度のみが考慮されるという意味でフィルタリングされます。 *name* は変数名です。

add_finite_strain_elasticity_brick(*mim*, *constitutive_law*, *varname*, *params*, *region=None*)

変数 *varname* を基準にしてモデルに非線形弾性項を追加します。 *lawname* は 'SaintVenant Kirchhoff' 、 'Mooney Rivlin' 、 'Neo Hookean' 、 'Ciarlet Geymonat' 、のいずれかです。対応するバージョンを使用するには、'Mooney Rivlin' および'Neo Hookean' 法の前に 'Compressible' または 'Incompressible' という語を付ける必要があります。これらの法則の圧縮性バージョンには、追加の材料係数が 1 つ必要です。

重要: 変数が 2 次元メッシュ上で定義されている場合、平面歪み近似が自動的に使用されます。 *params* は構成則のパラメータのベクトルです。長さは法則によります。これは、定数値の短いベクトル、または可変係数の有限要素法で記述されたベクトルフィールドです。'region' は項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。このブリックは、高水準の汎用アセンブリを使用します。モデル内のブリックのインデックスを返します。

add_finite_strain_elastoplasticity_brick(*mim*, *lawname*, *unknowns_type*, *varnames=None*, **args*)

概要: ind = Model.add_finite_strain_elastoplasticity_brick(self, MeshIm mim , string lawname, string unknowns_type [, string varnames, ...] [, string params, ...] [, int region = -1])

有限ひずみ弾塑性ブリックをモデルに追加します。現在のところ、サポートされている法則は、 *lawname* によって "Simo_Miehe" と定義されたものだけです。この法則は、 'DISPLACEMENT_AND_PLASTIC_MULTIPLIER' (整数値 1) または 'DISPLACEMENT_AND_PLASTIC_MULTIPLIER_AND_PRESSURE' (整数値 3) のいずれかに設定された *unknowns_type* によって定義される未知変数求解の可能性をサポートします。"Simo_Miehe" の法則では、モデル内で変数として定義する必要がある次の名前の集合を *varnames* と想定しています。

- 未知変数として定義されなければならない変位です。
- 可塑乗数も未知と定義されています。
- オプションで、'DISPLACEMENT_AND_PLASTIC_MULTIPLIER_AND_PRESSURE' の混合変位-圧力式の圧力変数を *unknowns_type* で指定します。
- 前の時間ステップでの塑性歪みを保持する (スカラー)fem_data または im_data フィールドの名前です。そして
- 直前の時間ステップでの右 Cauchy-Green テンソルの逆数の非反復成分をすべて保持する fem_data または im_data フィールドの名前です (平面歪み 2 次元問題では 4 要素ベクトル、3 次元問題では 6 要素ベクトルでなければなりません)。

"Simo_Miehe" 法はまた、次の 3 つのパラメータの集合をパラメータとしています。

- 初期体積弾性率 K の式です、
- 初期せん断弾性率 G の式です、
- 硬化変数 (降伏限界と硬化変数の値は、それぞれ適切な応力テンソルと歪テンソルの Frobenius ノルムと仮定します) の関数として降伏限界を表示するユーザ定義関数の名前です。

通常、 *region* は項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。モデル内のブリックインデックスを返します。

add_finite_strain_incompressibility_brick(*mim*, *varname*, *multname_pressure*, *region=None*)

有限ひずみ非圧縮性条件を (有限ひずみ弾性の場合) に追加します。*multname_pressure* は圧力を表す変数である。圧力を記述する有限要素法と主変数との間の入力条件を満

たす必要があることに注意してください。 *region* は項が追加されるオプションのメッシュ領域です。指定しない場合はメッシュ全体に追加されます。モデル内のブリックインデックスを返します。p が乗数で、u が変位の場合、"非線形非圧縮性ブリック" で、p*(1-Det(Id(meshdim)+Grad_u)) の高水準汎用アセンブリを追加すると等しいです。

add_generalized_Dirichlet_condition_with_Nitsche_method(*mim*, *varname*, *Neumannterm*, *gamma0name*, *region*, *theta=None*)

変数 *varname* とメッシュ領域 *region* に Dirichlet 条件を追加します。このバージョンはベクトルフィールド用です。これは @f$ Hu = r @f$ という条件を規定しており、ここで *H* は行列フィールドです。注意: 行列 H の固有値はすべて 1 または 0 である必要があります。領域は境界である必要があります。 *Neumannterm* は高水準汎用アセンブリ言語の表現として説明される Neumann 項 (Green 公式により得られる) の表現です。この項は Model.Neumann_term(varname, region) によって model に追加されます。すべてのボリュームブリックがモデルに追加されると。ディリクレ条件は Nitsche 法で処理されています。 *dataname* は Dirichlet 条件のオプションの右辺です。これは一定であるか、または有限要素法で記述できます。 *gamma0name* は Nitsche 法のパラメータです。 *theta* は正または負のスカラー値です。 *theta = 1* は *gamma0* が小さい場合に条件的に強制される標準的な対称法に相当する。 *theta = -1* は無条件に強制的な skew 対称法に対応します。 *theta = 0* は非線形問題に対しても Neumann 項の 2 次導関数を必要としない最も単純な方法である。 *Hname* は行列フィールド *H* に対応するデータである。定数行列であるか、スカラー関数で記述されている必要があります。モデル内のブリックのインデックスを返します。(このブリックは完全にはテストされていせん)。

add_generalized_Dirichlet_condition_with_multipliers(*mim*, *varname*, *mult_description*, *region*, *dataname*, *Hname*)

変数 *varname* とメッシュ領域 *region* に Dirichlet 条件を追加します。このバージョンはベクトルフィールド用です。ここで、 *H* は行列フィールドです。領域は境界である必要があります。Dirichlet 条件は *mult_description* によって記述される乗数変数で規定される。 *mult_description* が文字列の場合は、乗数 (これは、最初にモデルのメッシュ領域で

乗数変数として宣言する必要があります) に対応する変数名と見なされます。有限要素法 (mesh_fem オブジェクト) の場合は、乗数変数がモデルに追加され、この有限要素法 (メッシュ領域 *region* に制限され、最終的に他の乗数変数との自由度の競合が抑制されます) に基づいて構築されます。整数の場合は、乗数変数がモデルに追加され、その整数の次数の従来の有限要素に基づいて構築されます。 *dataname* は Dirichlet 条件の右側です。定数の場合もあれば、有限要素法で記述される場合もあります。スカラー値またはベクトル値で、Dirichlet 条件が指定されている変数によって異なります。 *Hname* は行列フィールド *H* に対応するデータです。モデル内のブリックのインデックスを返します。

add_generalized_Dirichlet_condition_with_penalization(*mim*, *varname*, *coeff*, *region*, *dataname*, *Hname*, *mf_mult=None*)

変数 *varname* とメッシュ領域 *region* に Dirichlet 条件を追加します。このバージョンはベクトルフィールド用です。ここで、 *H* は行列フィールドです。領域は境界である必要があります。ディリクレ状態はペナリゼーションとともに処方される。ペナルティ係数は本来 *coeff* であり、モデルのデータに追加されます。 *dataname* は Dirichlet 条件の右辺です。定数の場合もあれば、有限要素法で記述される場合もあります。スカラー値またはベクトル値で、Dirichlet 条件が指定されている変数によって異なります。 *Hname* は行列フィールド *H* に対応するデータです。定数行列であるか、スカラー関数で記述されている必要があります。 *mf_mult* はオプションのパラメータで、乗数空間を指定して Dirichlet 条件を弱めることができます。モデル内のブリックインデックスを返します。

add_generic_elliptic_brick(*mim*, *varname*, *dataname*, *region=None*)

変数 *varname* を基準にして、一般的な楕円項をモデルに追加します。楕円項の形状は変数とデータの両方に依存する。これは、以下の項に対応します。 $-\mathrm{div}(a\nabla u)$ ここで、 a はデータ、 u は変数です。データはスカラー、行列、または 4 次テンソルです。変数はベクトル値であってもなくてもかまいません。データがスカラーまたはマトリックスで、変数がベクトル値の場合、項はコンポーネント単位で追加されます。4 次のテンソルデータは、ベクトル値変数にのみ使用できます。データは一定であってもよいし、数分で記述してもよいです。もちろん、データが有限要素法 (テンソル場) で記述されるテンソルである場合、データは巨大なベクトルになる可能性があります。行列/テンソルの成分は (blas との互換性から) データベクトルの Fortran(列方向) 次数で保存されなければなりません。与えられた行列/テンソルの (仮定されている) 対称性は検証されません。これがベクトル値の変数である場合、楕円項はコンポーネントごとに追加されます。 *region* は、項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。高水準汎用アセンブリ言語を使用する実数バージョンでは、モデル変数にも依存しますが、(例えば "1" 、 "sin(X(1))" 、"Norm(u)" など) は高水準汎用アセンブリ言語の任意の

正規表現にすることができます。モデル内のブリックインデックスを返します。

add_im_data(*name*, *mimd*)

MeshImd にリンクされたモデルにデータセットを追加します。 *name* はデータ名です。

add_initialized_data(*name*, *V*, *sizes=None*)

初期化された固定サイズデータをモデルに追加します。 *sizes* はオプションのパラメータで、データのフォーマットを記述する整数またはベクトルの次元です。デフォルトでは、データはベクトルフィールド b とみなされます。 *name* はデータ名で、 *V* はデータの値です。

add_initialized_fem_data(*name*, *mf*, *V*, *sizes=None*)

MeshFem にリンクされたモデルにデータを追加します。 *name* はデータ名です。データは *V* で初期化されます。データはスカラーまたはベクトルフィールドです。 *sizes* はオプションのパラメータであり、 'mf'に対する補完次元の整数またはベクトルです。

add_integral_contact_between_nonmatching_meshes_brick(*mim*, *varname_u1*, *varname_u2*, *multname*, *dataname_r*, *dataname_friction_coeff=None*, **args*)

概要: ind = Model.add_integral_contact_between_nonmatching_meshes_brick(self, MeshIm mim, string varname_u1, string varname_u2, string multname, string dataname_r [, string dataname_friction_coeff], int region1, int region2 [, int option [, string dataname_alpha [, string dataname_wt1 , string dataname_wt2]]])

摩擦条件の有無にかかわらず、一致しないメッシュ間に接触をモデルに追加します。このブリックは、完全な方法で定義された接触を追加します。これは, 連続レベルで定義された拡張 Lagrangian 定式化 (Getfem ユーザマニュアルを参照) の直接近似です。この利点は、スケーラビリティの向上です。Newton 法の反復回数は、メッシュサイズに多少依存しません。条件は、 *region1* および *region2* に対応する境界上の変数 *varname_u1* および *varname_u2* に適用されます。 *multname* は、摩擦がない場合の接触応力と、摩擦がある場合の接触および摩擦応力を表す有限要素法の変数である必要があります。 *multname* と 'varname_u1' と *varname_u2* の間の inf-sup 条件が必要です。拡張パラメータ *dataname_r* は許容値の範囲内で選択する必要があります。オプションのパラメータ *dataname_friction_coeff* は摩擦係数で、一定であるか、または *varname_u1* と同じメッシュ上の有限要素法で定義できます。 *option* の値は、非対称 Alart-Curnier 拡張 Lagrangian 法では 1、対称的なものでは 2、追加の拡張を伴う非対称 Alart-Curnier 法では 3、新しい非対称法では 4 となります。デフォルト値は 1 です。摩擦がある場合、 *dataname_alpha* 、

dataname_wt1 および *dataname_wt2* は、摩擦の進展を解くためのオプションパラメータです。

add_integral_contact_with_rigid_obstacle_brick(*mim*, *varname_u*, *multname*, *dataname_obstacle*, *dataname_r*, *dataname_friction_coeff=None*, **args*)

概要: ind = Model.add_integral_contact_with_rigid_obstacle_brick(self, MeshIm mim, string varname_u, string multname, string dataname_obstacle, string dataname_r [, string dataname_friction_coeff], int region [, int option [, string dataname_alpha [, string dataname_wt [, string dataname_gamma [, string dataname_vt]]]]])

摩擦条件の有無にかかわらず、剛体障害物のある接触をモデルに追加します。このブリックは、完全な方法で定義された接触を追加します。これは連続レベルで定義された拡張ラグランジュ定式化 (Getfem ユーザマニュアルを参照) の直接近似である。利点は拡張性が高いことです。ニュートン法の反復回数は、メッシュサイズに多少依存しません。接触条件は *region* に対応する境界上の変数 *varname_u* に適用されます。剛体障害物は、データ *dataname_obstacle* が障害物までの符号付き距離 (有限要素法による補間) で記述されるべきである。 *multname* は接触応力を表す有限要素法変数でなければなりません。 *multname* と *varname_u* の間の inf-sup 条件が必要です。拡張パラメータ *dataname_r* は、許容値の範囲内で選択する必要があります。オプションのパラメータ *dataname_friction_coeff* は摩擦係数で、一定にすることも、有限要素法で定義することもできます。 *option* の可能な値は、非対称 Alart-Curnier 拡張 Lagrangian 法では 1、対称的なものでは 2、追加の拡張を伴う非対称 Alart-Curnier 法では 3、新しい非対称法では 4 です。デフォルト値は 1 です。摩擦がある場合、 *dataname_alpha* と *dataname_wt* は摩擦の進化を解決するためのオプションパラメータです。*dataname_gamma* および *dataname_vt* は、パラメータに依存するすべり速度を摩擦条件に追加するためのオプションデータを表します。

add_integral_large_sliding_contact_brick_raytracing(*dataname_r*, *release_distance*, *dataname_fr=None*, **args*)

概要: ind = Model.add_integral_large_sliding_contact_brick_raytracing(self, string dataname_r, scalar release_distance, [, string dataname_fr[, string dataname_alpha[, int version]]])

摩擦ブリックとの有限すべり接触をモデルに追加します。このブリックは自己接触、複数の変形可能体間の接触及び剛体障害物との接触に対処できます。高水準汎用アセンブリを使用します。モデルに raytracing_interpolate_transformation オブジェクトを追加します。

スレーブ境界ごとに乗数変数を定義する必要があります。リリース距離は慎重に決定すべきです (一般に平均要素サイズの数倍で、物体の厚さよりも薄いです)。最初は、ブリックは接触境界なしで追加されます。接触境界と剛体には、特別な機能が追加されています。*version* は、非対称バージョンでは 0(デフォルト値)、より対称なバージョンでは 1(摩擦がなくても完全に対称的でない) です。

add_interpolate_transformation_from_expression(*transname*, *source_mesh*, *target_mesh*, *expr*)

メッシュ *source_mesh* から式 *expr* で指定されるメッシュ *target_mesh* への変換をモデルに追加します。この式は、モデルの変数を含む高水準汎用アセンブリ式に対応します。注意: 接線システムの計算では、使用した変数による変換の導関数が考慮されます。ただし、2 次微分は実装されていないため、ポテンシャルの定義ではこのような変換は許されていません。

add_isotropic_linearized_elasticity_brick(*mim*, *varname*, *dataname_lambda*, *dataname_mu*, *region=None*)

等方性線形化弾性項を、変数 *varname* に対して相対的にモデルに追加します。*dataname_lambda* と *dataname_mu* には、Lame 係数を含める必要があります。 *region* は項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。モデル内のブリックインデックスを返します。

add_isotropic_linearized_elasticity_brick_pstrain(*mim*, *varname*, *data_E*, *data_nu*, *region=None*)

等方性線形化弾性項を、変数 *varname* に対してモデルに追加します。 *data_E* と *data_nu* には、それぞれ Young 率と Poisson 比を含める必要があります。 *region* はオプションの項が追加されるメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。2 次元メッシュでは、この項は単純な歪み近似と相関します。3 次元メッシュでは、標準モデルに対応します。モデル内のブリックのインデックスを返します。

add_isotropic_linearized_elasticity_brick_pstress(*mim*, *varname*, *data_E*, *data_nu*, *region=None*)

等方性線形化弾性項を、変数 *varname* に対してモデルに追加します。 *data_E* と *data_nu* には、それぞれ Young 率と Poisson 比を含める必要があります。 *region* は項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。2 次元メッシュでは、この項は単純応力近似に対応します。3 次元メッシュでは、標準モデルに対応します。モデル内のブリックインデックスを返します。

add_linear_generic_assembly_brick(*mim*, *expression*, *region=None*, **args*)

概要: ind = Model.add_linear_generic_assembly_brick(self, MeshIm mim, string expression[, int region[, int is_symmetric[, int is_coercive]]])

非推奨。代わりに Model.add_linear_term() を使用します。

add_linear_incompressibility_brick(*mim*, *varname*, *multname_pressure*, *region=None*, **args*)

概要: ind = Model.add_linear_incompressibility_brick(self, MeshIm mim, string varname, string multname_pressure[, int region[, string dataexpr_coeff]])

variable に線形非圧縮性条件を追加します。 *multname_pressure* は圧力を表す変数です。圧力を記述する有限要素法と主変数との間の入力条件を満たす必要があることに注意してください。 *region* は項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。 *dataexpr_coeff* は、たとえば、非圧縮性に近い弾性に対するオプションのペナルティ係数です。この場合、それは Lame 係数の逆数である λ です。モデル内のブリックインデックスを返します。

add_linear_term(*mim*, *expression*, *region=None*, **args*)

概要: ind = Model.add_linear_term(self, MeshIm mim, string expression[, int region[, int is_symmetric[, int is_coercive]]])

アセンブリ文字列 *expr* で指定された行列項を追加します。この行列は、領域 *region* で積分法 *mim* を使用して組み立てられます。線形であると仮定すると、行列項のみが考慮されます。項を非線形ではなく線形と宣言する利点は、項が 1 回だけ組み立てられ、残差にアセンブリが必要ないことです。式にいくつかの変数が含まれ、式がポテンシャルまたは一次 (すなわち、弱形式の微分ではなく、弱形式を記述する) 場合、式はすべての変数に関して微分することに注意してください。項が対称か、強制でないかどうかを指定できます。よくわからない場合は、対称ではなく、強制的ではないと宣言した方がよいでしょう。しかし、一部のソルバー (例えば共役勾配) は非強制問題には使用できません。 *brickname* はブリックのオプション名です。

add_linear_twodomain_term(*mim*, *expression*, *region*, *secondary_domain*, *is_symmetric=None*, **args*)

概要: ind = Model.add_linear_twodomain_term(self, MeshIm mim, string expression, int region, string secondary_domain[, int is_symmetric[, int is_coercive]])

Model のような弱形式言語式で与えられる線形項を追加します。add_linear_term() ではなく、二つのドメインの直接の積上での統合の場合には、最初に ‘‘mim” と ‘‘region” で指定されたものと ‘‘secondary_domain” で指定されたものを最初にモデルに宣言しなければなりません。

add_macro(*name*, *expr*)

汎用アセンブリ言語の新しいマクロを定義します。名前にはパラメータが含まれます。例えば、name=’sp(a,b)’ 、expr=’a.b’ は有効な定義です。パラメータのないマクロも定義でき

ます。たとえば、name='x1'、expr='X[1]' は有効です。 name='grad(u)'、expr='Grad_u' という形式も使用できますが、この場合、パラメータ 'u' はマクロの使用時にのみ変数名として許可されます。マクロは、キーワード 'Def' を使用してアセンブリ 文字列内に直接定義できます。

add_mass_brick(*mim*, *varname*, *dataexpr_rho=None*, **args*)

概要: ind = Model.add_mass_brick(self, MeshIm mim, string varname[, string dataexpr_rho[, int region]])

変数 *varname* に対して質量項をモデルに追加します。指定された場合、データ *dataexpr_rho* は密度 (省略した場合は 1) です。 *region* は、項が追加されるオプションのメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。モデル内のブリックのインデックスを返します。

add_master_contact_boundary_to_biased_Nitsche_large_sliding_contact_brick(*in*
m
re
gi
di
na
wr

摩擦ブリックと既存のバイアスされた Nitsche の有限スライド 接触にマスター接触境界を追加します。

add_master_contact_boundary_to_large_sliding_contact_brick(*indbrick*, *mim*, *region*, *dispname*, *wname=None*)

摩擦ブリックを持つ既存の大きなスライド 接触にマスター接触境界を追加します。

add_master_contact_boundary_to_projection_transformation(*transname*, *m*, *dispname*, *region*)

特定の境界 *region* 上に対応する変位変数 *dispname* を持つマスター接触境界を、'transname'という既存の投影補間変換に追加します。

add_master_contact_boundary_to_raytracing_transformation(*transname*, *m*, *dispname*, *region*)

特定の境界 *region* 上の対応する変位変数 *dispname* を持つマスター接触境界を *transname* という既存のレイトレーシング補間変換に追加します。

add_master_slave_contact_boundary_to_large_sliding_contact_brick(*indbrick*, *mim*, *region*, *dispname*, *lambdaname*, *wname=None*)

接触境界を、マスターでもスレーブでもある摩擦レンガの既存の有限スライド 接触に追加します (自己接触は許容します)。

add_multiplier(*name*, *mf*, *primalname*, *mim=None*, *region=None*)

有限要素法にリンクされた特定の変数を、原始変数に対する乗数として追加します。自由度は `gmm::range_basis` 関数でフィルタリングされ、乗数と原始変数をリンクするモデルの項に適用されます。これは、主変数に対する線形独立制約のみを保持するためです。境界乗数用に最適化されています。

add_nodal_contact_between_nonmatching_meshes_brick(*mim1*, *mim2=None*, **args*)

概要: ind = Model.add_nodal_contact_between_nonmatching_meshes_brick(self, MeshIm mim1[, MeshIm mim2], string varname_u1[, string varname_u2], string multname_n[, string multname_t], string dataname_r[, string dataname_fr], int rg1, int rg2[, int slave1, int slave2, int augmented_version])

1 つまたは 2 つの弾性ボディの 2 つの面の間に、摩擦条件の有無にかかわらず接触を追加します。条件は、変数 *varname_u1* または変数 *varname_u1* と *varname_u2* に適用されます。これは、1 つまたは 2 つの個別の変位フィールドが指定されているかどうかによって異なります。整数 *rg1* および *rg2* は、互いに接触すると予想される領域を表す。単一変位変数の場合、 *rg1* と *rg2* の両方で定義された領域は、変数 'varname_u1' を参照します。2 つの変位変数の場合、*rg1* は *varname_u1* を参照し、*rg2* は *varname_u2* を参照します。*multname_t* は、*rg1* と 'rg2' で定義されている領域のうち、 "slaves" とされている領域の自由度をサイズとする固定サイズ変数です。接触等価節点垂直力を表します。*multname_t*

は、qdim-1 で乗算された *multname_n* のサイズに対応するサイズの固定サイズ変数でなければなりません。接触等価節点正接 (摩擦の) 力を表します。拡張パラメータ *r* は、許容値 (弾性体の Young 率に近い値。Getfem のユーザードキュメントを参照) の範囲内で選択されるべきです。パラメータ *fr* に格納される摩擦係数は、単一の値または *multname_n* と同じサイズのベクトルのいずれかです。オプションのパラメータ *slave1* と *slave2* は、 *rg1* と *rg2* で定義された領域が "slaves" とみなされるかどうかを宣言します。デフォルトでは、 *slave1* は true、 *slave2* は false です。つまり、 'rg1' はスレーブ面を含み、マスタ面を 'rg2' とします。 *slave1* と 'slave2' のどちらか一方だけを true に設定するのが好ましいです。パラメータ *augmented_version* は拡大戦略を示しｍす: 1 は非対称の Alart-Curnier 拡張 Lagrangian、2 は対称のもの (接触と Coulomb 摩擦の結合を除きます)、3 は新しい非対称法です。基本的に、このブリックは行列 BN と BT とベクトル gap と alpha を計算し、基本的な接触ブリックを呼び出します。

add_nodal_contact_with_rigid_obstacle_brick(*mim*, *varname_u*, *multname_n*, *multname_t=None*, **args*)

概要: ind = Model.add_nodal_contact_with_rigid_obstacle_brick(self, MeshIm mim, string varname_u, string multname_n[, string multname_t], string dataname_r[, string dataname_friction_coeff], int region, string obstacle[, int augmented_version])

摩擦条件の有無にかかわらず、剛体障害物のある接触をモデルに追加します。条件は、*region* に対応する境界上の変数 *varname_u* に適用されます。剛障害物は、障害物への符号付き距離である文字列 *obstacle* で記述します。この文字列は、座標が 2 次元では 'x'、'y' 、3 次元では'x'、 'y'、 'z' の式である必要があります。たとえば、障害物が $z \leq 0$ に対応する場合、対応する符号付き距離は単に "z" になります。 *multname_n* はサイズが境界 *region* の自由度数である固定サイズ変数でなければなりません。これは接触等価節点力を表します。摩擦条件を追加するには、 *multname_t* および *dataname_friction_coeff* パラメータを追加する必要があります。*multname_t* は固定サイズの変数でなければならず、そのサイズは境界 'region' 上の自由度の数に $d-1$ を乗じたものである。ここで d は領域の次元です。これは摩擦相当節点力を表します。拡張パラメータ *r* は許容値 (弾性体の Young 率に近い値。Getfem のユーザードキュメントを参照) の範囲内で選択される必要があります。*dataname_friction_coeff* は摩擦係数です。各接触節点の摩擦係数を表すスカラー値または値のベクトルです。パラメータ *augmented_version* は拡大戦略を示します: 1 は非対称の Alart-Curnier 拡張 Lagrangian、2 は対称 (接触と Coulomb 摩擦の結合を除く)、3 は新しい非対称方法です。基本的に、このブリックは行列 BN およびベクトル gap および alpha を計算し、基本接触ブリックを呼び出します。

add_nonlinear_elasticity_brick(*mim*, *varname*, *constitutive_law*, *dataname*, *region=None*)

変数 *varname* (deprecated brick、代わりに add_finite_strain_elasticity を使用してください) に関する非線形弾性項をモデルに追加します。*lawname* は構成則であり 'SaintVenant Kirchhoff' 、 'Mooney Rivlin' 、 'neo Hookean' 、 'Ciarlet Geymonat' または 'generalized

Blatz Ko' のいずれかになります。 'Mooney Rivlin' および 'neo Hookean' 則の名前の前に 'compressible' または 'incompressible' を付けると、対応するバージョンが使用されます。これらの法則の圧縮性バージョンには、追加の材料係数が 1 つ必要です。既定では、'Mooney Rivlin' 則の非圧縮性バージョンと 'neo Hookean' 則の圧縮性バージョンが考慮されます。一般に 'neo Hookean' は'Mooney Rivlin' 則の特殊なケースであり、一つの係数を少なくする必要があります。重要:変数が 2 次元メッシュ上で定義されている場合、平面歪み近似が自動的に使用されます。 *dataname* は構成則のパラメータのベクトルです。長さは法則によります。これは、定数値の短いベクトル、または可変係数の有限要素法で記述されたベクトルフィールドです。 *region* はオプションの項が追加されるメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。このブリックは、低水準汎用アセンブリを使用します。モデル内のブリックのインデックスを返します。

add_nonlinear_generic_assembly_brick(*mim*, *expression*, *region=None*, **args*)

概要: ind = Model.add_nonlinear_generic_assembly_brick(self, MeshIm mim, string expression[, int region[, int is_symmetric[, int is_coercive]]])

非推奨。 Model.add_nonlinear_term() を使用します。

add_nonlinear_incompressibility_brick(*mim*, *varname*, *multname_pressure*, *region=None*)

非線形非圧縮性条件を (有限歪み弾性の場合) に追加します。 *multname_pressure* は圧力を表す変数です。圧力を記述する有限要素法と主変数との間の入力条件を満たす必要があることに注意してください。 *region* はオプションの項が追加されるメッシュ領域です。指定しない場合は、メッシュ全体に追加されます。モデル内のブリックインデックスを返します。

add_nonlinear_term(*mim*, *expression*, *region=None*, **args*)

概要: ind = Model.add_nonlinear_term(self, MeshIm mim, string expression[, int region[, int is_symmetric[, int is_coercive]]])

アセンブリ文字列 *expr* で指定された非線形項を追加します。この非線形項は、領域 *region* と積分法 *mim* で組み立てられます。この表現は、ポテンシャルまたは弱形式を表すことができます。2 次項 (すなわち、2 次試験関数 Test2 を含みます) は使用できません。項が対称か、強制かどうかを指定できます。よくわからない場合は、対称ではなく、強制ではないと宣言した方がよいでしょう。しかし、一部のソルバー (例えば共役勾配) は非強制問題には使用できません。 *brickname* はブリックのオプション名です。

add_nonlinear_twodomain_term(*mim*, *expression*, *region*, *secondary_domain*, *is_symmetric=None*, **args*)

概要: ind = Model.add_nonlinear_twodomain_term(self, MeshIm mim, string expression, int region, string secondary_domain[, int is_symmetric[, int is_coercive]])

弱形式言語の式で与えられる非線形項を追加します。 Model.add_nonlinear_term() のよう

ではなく、二つのドメインの直積上での積分の場合には、最初に mim と region で指定されたものと secondary_domain で指定されたものを最初にモデルに宣言しなければなりません。

add_nonmatching_meshes_contact_brick(*mim1*, *mim2=None*, **args*)

概要: ind = Model.add_nonmatching_meshes_contact_brick(self, MeshIm mim1[, MeshIm mim2], string varname_u1[, string varname_u2], string multname_n[, string multname_t], string dataname_r[, string dataname_fr], int rg1, int rg2[, int slave1, int slave2, int augmented_version])

非推奨機能です。代わりに 'add nodal contact between nonmatching meshes brick' を使用してください。

add_normal_Dirichlet_condition_with_Nitsche_method(*mim*, *varname*, *Neumannterm*, *gamma0name*, *region*, *theta=None*, **args*)

概要: ind = Model.add_normal_Dirichlet_condition_with_Nitsche_method(self, MeshIm mim, string varname, string Neumannterm, string gamma0name, int region[, scalar theta][, string dataname])

ベクトル (またはテンソル) 値変数 *varname* の法線コンポーネントとメッシュ領域 *region* に Dirichlet 条件を追加します。この領域は境界である必要があります。 *Neumannterm* は、高水準汎用アセンブリ言語の表現として説明される Neumann 項 (Green 公式により得られる) の表現である。この項は Model.Neumann_term(varname, region) によって得られます。すべてのボリュームブリックがモデルに追加されると、ディリクレ条件は Nitsche の方法で処方される。 *dataname* は Dirichlet 条件のオプションの右辺です。これは一定であるか、または有限要素法で記述できます。 *gamma0name* は Nitsche 法パラメータです。 *theta* は正または負のスカラー値です。 *theta = 1* は *gamma0* が小さい場合に条件的に強制される標準的な対称法に相当する。 *theta = -1* は無条件に強制的な skew 対称法に対応します。 *theta = 0* は非線形問題に対しても Neumann 項の 2 次導関数を必要としない最も単純な方法である。モデル内のブリックのインデックスを返します。(このブリックは完全にはテストされていません)

add_normal_Dirichlet_condition_with_multipliers(*mim*, *varname*, *mult_description*, *region*, *dataname=None*)

ベクトル (またはテンソル) 値変数 *varname* の法線コンポーネントとメッシュ領域 *region* に Dirichlet 条件を追加します。この領域は境界である必要があります。Dirichlet 条件は *mult_description* によって記述される乗数変数で規定される。 *mult_description* が文字列

の場合は、乗数 (これは、最初にモデルのメッシュ領域で乗数変数として宣言する必要があります) に対応する変数名と見なされます。有限要素法 (mesh_fem オブジェクト) の場合は、乗数変数がモデルに追加され、この有限要素法 (メッシュ領域 *region* に制限され、最終的に他の乗数変数との自由度の競合が抑制されます) に基づいて構築されます。整数の場合は、乗数変数がモデルに追加され、その整数の次数の従来の有限要素に基づいて構築されます。 *dataname* は Dirichlet 条件のオプションの右辺です。定数の場合もあれば、有限要素法で記述される場合もあります。スカラー値またはベクトル値で、Dirichlet 条件が指定されている変数によって決まります (変数がベクトル値の場合はスカラー、テンソル値の場合はベクトルです)。モデル内のブリックのインデックスを返します。

add_normal_Dirichlet_condition_with_penalization(*mim*, *varname*, *coeff*, *region*, *dataname=None*, *mf_mult=None*)

ベクトル (またはテンソル) 値変数 *varname* の法線コンポーネントとメッシュ領域 *region* に Dirichlet 条件を追加します。この領域は境界である必要があります。Dirichlet 条件はペナリゼーションとともに処理される。ペナルティ係数は最初は *coeff* であり、モデルのデータに追加されます。 *dataname* は Dirichlet 条件のオプションの右辺です。定数の場合もあれば、有限要素法で記述される場合もあります。スカラー値またはベクトル値で、Dirichlet 条件が指定されている変数によって決まります (変数がベクトル値の場合はスカラー、テンソル値の場合はベクトル)。 *mf_mult* はオプションのパラメータで、乗数空間を指定して Dirichlet 条件を弱めることができます。モデル内のブリックのインデックスを返します。

add_normal_derivative_Dirichlet_condition_with_multipliers(*mim*, *varname*, *mult_description*, *region*, *dataname=None*, *R_must_be_derivated=N*

変数 *varname* の法線導関数とメッシュ領域 *region* (境界である必要がありますに Dirichlet 条件を追加します。一般的な形式は以下の通りである:math:*int partial_n u(x)v(x) = int r(x)v(x) forall v* ここで、 $r(x)$ は Dirichlet 条件 (均一条件の場合は 0) の右辺であり、 v は *mult_description* で定義される乗数の空間にある。 *mult_description* が文字列の場合は、乗数 (これは、最初にモデルのメッシュ領域で乗数変数として宣言する必要があります) に対応する変数名と見なされます。有限要素法 (mesh_fem オブジェクト) の場合は、乗数変数がモデルに追加され、この有限要素法 (メッシュ領域 *region* に制限され、最終的に他の乗数変数との自由度の競合が抑制されます) に基づいて構築されます。整数の場合は、乗数変数がモデルに追加され、その整数の次数の従来の有限要素に基づいて構築

されます。 *dataname* は Dirichlet 条件の右辺を表すオプションのパラメータです。もし *R_must_be_derivated* が *true* に設定されていれば、 *dataname* の正規導関数が考慮されます。モデル内のブリックインデックスを返します。

add_normal_derivative_Dirichlet_condition_with_penalization(*mim*, *varname*, *coeff*, *region*, *dataname=None*, *R_must_be_derivated=*

変数 *varname* の法線導関数とメッシュ領域 *region* (境界である必要がありますに Dirichlet 条件を追加します。一般的な形式は以下の通りである $\int \partial_n u(x)v(x) = \int r(x)v(x)\forall v$ ここで $r(x)$ は Dirichlet 条件 (均一条件の場合は 0) の右辺です。ペナルティ係数は最初は *coeff* であり、モデルのデータに追加されます。これは、 Model.change_penalization_coeff() コマンドで変更できます。 *dataname* は Dirichlet 条件の右辺を表すオプションのパラメータです。もし *R_must_be_derivated* が *true* に設定されていれば、 *dataname* の正規導関数が考慮されます。モデル内のブリックインデックスを返します。

add_normal_derivative_source_term_brick(*mim*, *varname*, *dataname*, *region*)

変数 *varname* とメッシュ領域 *region* に通常の微分ソース項ブリック $F = \int b.\partial_n v$ を追加します。

線形システムの右辺を更新します。 *dataname* は *b* を表し、 *varname* は *v* を表します。モデル内のブリックインデックスを返します。

add_normal_source_term_brick(*mim*, *varname*, *dataname*, *region*)

境界 *region* 上の変数 *varname* にソース項を追加します。この領域は境界である必要があります。ソースタームは、データ *dataepxpr* によって表され、これは、高水準汎用アセンブリ言語 (ただし、モデルの宣言されたデータでなければならない複素数バージョンは除きます) の任意の正規表現であり得る。境界への外向き法線単位ベクトルを持つスカラー積が実行されます。このブリックの主な目的は事前処理として法線を持つスカラー積を実行せずにベクトルデータで Neumann 条件を表現することです。モデル内のブリックインデックスを返します。

add_penalized_contact_between_nonmatching_meshes_brick(*mim*, *varname_u1*, *varname_u2*, *dataname_r*, *dataname_coeff=None*, **args*)

概要: ind = Model.add_penalized_contact_between_nonmatching_meshes_brick(self, MeshIm mim, string varname_u1, string varname_u2, string dataname_r [, string dataname_coeff], int region1, int region2 [, int option [, string dataname_lambda, [, string dataname_alpha [, string dataname_wt1, string dataname_wt2]]]])

モデルに、一致しないメッシュ間の摩擦の有無にかかわらずペナルティ接触条件を追加します。条件は、 *region1* および *region2* に対応する境界上の変数 *varname_u1* および *varname_u2* に適用されます。ペナルティパラメータ *dataname_r* は、近似の非貫通条件および摩擦条件を規定するのに十分な大きさに選択する必要がありますが、接線システムの条件を過度に悪化させないようにするには大きすぎません。オプションのパラメータ *dataname_friction_coeff* は摩擦係数で、一定であるか、または *varname_u1* と同じメッシュ上の有限要素法で定義できます。 *dataname_lambda* は、option が 2 の場合に使用されるオプションのパラメータです。その場合、ペナルティ項は lambda 単位でシフトされます (これは、対応する拡張 Lagrangian 定式化上での Uzawa アルゴリズムの使用を可能にします)。接触摩擦の場合、 *dataname_alpha* 、 *dataname_wt1* および *dataname_wt2* は進展摩擦の問題を解決するためのオプションのパラメータです。

add_penalized_contact_with_rigid_obstacle_brick(*mim*, *varname_u*, *dataname_obstacle*, *dataname_r*, *dataname_coeff=None*, **args*)

概要: ind = Model.add_penalized_contact_with_rigid_obstacle_brick(self, MeshIm mim, string varname_u, string dataname_obstacle, string dataname_r [, string dataname_coeff], int region [, int option, string dataname_lambda, [, string dataname_alpha [, string dataname_wt]]])

摩擦条件の有無にかかわらず、剛体障害物のペナルティ接触をモデルに追加します。条件は *region* に対応する境界上の変数 *varname_u* に適用されます。剛体障害物はデータ *dataname_obstacle* が障害物までの符号付き距離 (有限要素法による補間) で記述されるべきである。ペナルティパラメータ *dataname_r* は、近似の非貫通条件および摩擦条件を規定するのに十分な大きさに選択する必要がありますが、接線システムの条件を過度に悪化させないようにするには大きすぎません。 *dataname_lambda* はオプションが 2 の場合に使用されるオプションのパラメータです。その場合、ペナルティ項は lambda 単位でシフトされます (これは、対応する拡張 Lagrangian 定式化上での Uzawa アルゴリズムの使用

を可能にします)。

add_pointwise_constraints_with_given_multipliers(*varname*, *multname*, *dataname_pt*, *dataname_unitv=None*, **args*)

概要: ind = Model.add_pointwise_constraints_with_given_multipliers(self, string varname, string multname, string dataname_pt[, string dataname_unitv] [, string dataname_val])

与えられた乗数 *multname* を使用して、変数 *varname* の各点に制約を追加します。条件は、データ *dataname_pt* に指定された点の集合で規定され、その寸法は点の数とメッシュの寸法の積です。乗数変数は、ポイント数のサイズを固定サイズ変数にする必要があります。変数がベクトルフィールドを表す場合には、データ *dataname_unitv* を与えなければなりません。これは、次元のベクトルを表すものであり、点の数と単位ベクトルを格納すべきベクトルフィールドの次元を乗算したものです。この場合、指定拘束は、対応する点における変数と対応する単位ベクトルとのスカラー積である。オプションのデータ *dataname_val* は異なる点で規定される値のベクトルです。このブリックは Neumann 問題における剛体変位を消滅させるために特別に設計されています。モデル内のブリックのインデックスを返します。

add_pointwise_constraints_with_multipliers(*varname*, *dataname_pt*, *dataname_unitv=None*, **args*)

概要: ind = Model.add_pointwise_constraints_with_multipliers(self, string varname, string dataname_pt[, string dataname_unitv] [, string dataname_val])

multiplier を使用して、変数 *varname* にポイントワイズ制約を追加します。乗数変数が自動的にモデルに追加されます。条件は、データ *dataname_pt* に指定された点のセットで規定され、その寸法は点の数とメッシュ次元の積である。変数がベクトルフィールドを表す場合には、データ *dataname_unitv* を与えなければならない。これは、次元のベクトルを表すものであり、点の数と単位ベクトルを格納すべきベクトルフィールドの次元を乗算したものである。この場合、指定拘束は、対応する点における変数と対応する単位ベクトルとのスカラー積である。オプションのデータ *dataname_val* は、異なる点で規定される値のベクトルです。このブリックは、Neumann 問題における剛体変位を消滅させるために特別に設計されている。モデル内のブリックのインデックスを返します。

add_pointwise_constraints_with_penalization(*varname*, *coeff*, *dataname_pt*, *dataname_unitv=None*, **args*)

概要: ind = Model.add_pointwise_constraints_with_penalization(self, string varname, scalar coeff, string dataname_pt[, string dataname_unitv] [, string dataname_val])

ペナルティにより、変数 *varname* に点ごとの制約を追加します。ペナルティ係数は最初は *penalization_coeff* で、モデルのデータに追加されます。条件は、データ *dataname_pt* に指定された点の集合で規定され、その次元は点の数とメッシュの次元の積です。変数がベクトルフィールドを表す場合には、データ *dataname_unitv* を与えなければなりません。これは、次元のベクトルを表すものであり、点の数と単位ベクトルを格納すべきベクトルフィールドの次元を乗算したものです。この場合、指定拘束は、対応する点における変数と対応する単位ベクトルとのスカラー積です。オプションのデータ *dataname_val* は異なる点で規定される値のベクトルです。このブリックは Neumann 問題における剛体変位を消滅させるために特別に設計されています。モデル内のブリックのインデックスを返します。

add_projection_transformation(*transname*, *release_distance*)

transname という名前の投影補間変換を、汎用アセンブリブリックで使用するモデルに追加します。注意: 現時点では、モデルの計算で変換の導関数は考慮されません。

add_raytracing_transformation(*transname*, *release_distance*)

汎用アセンブリブリックで使用されるモデルに *transname* というレイトレーシング補間変換を追加します。注意: 現時点ではモデルの計算で変換の導関数は考慮されません。

add_rigid_obstacle_to_Nitsche_large_sliding_contact_brick(*indbrick*, *expr*, *N*)

摩擦ブリックとの既存の有限すべり接触に、硬質の障害物を追加します。 *expr* は高水準汎用アセンブリ言語 (ここで、 *x* はメッシュの現在の点です) を使用した式であり、障害物までの符号付き距離である必要があります。 *N* はメッシュの次元です。

add_rigid_obstacle_to_large_sliding_contact_brick(*indbrick*, *expr*, *N*)

摩擦ブリックとの既存の有限すべり接触に、硬質の障害物を追加します。 *expr* は高水準汎用アセンブリ言語 (ここで、 *x* はメッシュの現在の点です) を使用した式であり、障害物までの符号付き距離である必要があります。 *N* はメッシュの次元です。

add_rigid_obstacle_to_projection_transformation(*transname*, *expr*, *N*)

ジオメトリが高水準汎用アセンブリ式 *expr* の 0 レベルセットに対応する剛体障害を *transname* という既存の投影補間変換に追加します。

add_rigid_obstacle_to_raytracing_transformation(*transname*, *expr*, *N*)

ジオメトリが上位レベルの汎用アセンブリ式 *expr* の 0 レベルセットに対応する剛体な障害物を ‘transname‘という既存のレイトレーシング補間変換に追加します。

add_slave_contact_boundary_to_biased_Nitsche_large_sliding_contact_brick(*indb*
mim
re-
gion
disp
nam
lam
dan
wna

摩擦ブリックとの既存のバイアスされた Nitsche の有限滑り 接触にスレーブ接触境界を追加します。

add_slave_contact_boundary_to_large_sliding_contact_brick(*indbrick*, *mim*, *region*, *dispname*, *lambdaname*, *wname=None*)

摩擦ブリックを持つ既存の有限滑り 接触にスレーブ接触境界を追加します。

add_slave_contact_boundary_to_projection_transformation(*transname*, *m*, *dispname*, *region*)

特定の境界 *region* 上の対応する変位変数 *dispname* を持つスレーブ接触境界を‘transname‘という既存の投影補間変換に追加します。

add_slave_contact_boundary_to_raytracing_transformation(*transname*, *m*, *dispname*, *region*)

特定の境界 *region* 上の対応するディスプレイスメント変数 *dispname* を持つスレーブコンタクト境界を、 *transname* という既存のレイトレーシング補間変換に追加します。

add_small_strain_elastoplasticity_brick(*mim*, *lawname*, *unknowns_type*, *varnames=None*, **args*)

概要: ind = Model.add_small_strain_elastoplasticity_brick(self, MeshIm mim, string lawname, string unknowns_type [, string varnames, ...] [, string params, ...] [, string theta = ’1’ [, string dt = ’timestep’]] [, int region = -1])

モデル *M* に微小歪み塑性項を追加します。これは、微小ひずみ塑性の主要な GetFEM++

ブリックです。 *lawname* は実行された塑性法則の名前であり、 *unknowns_type* は、塑性乗数が問題の未知である場合の離散化、または (リターンマッピング法) が次の反復のために格納されるモデルのデータのみの選択を示します。どちらの場合も、乗数は格納されます。*varnames* は、プラスチック則に依存する長さの変数名とデータ名の集合である (少なくとも変位、塑性乗数、塑性歪み)。*params* はパラメータの式のリストです (少なくとも弾性係数と降伏応力)。これらの式には、モデルのデータ名 (または変数名) を使用できますが、高水準アセンブリ言語 (例えば *'1/2'*、 *'2+sin(X[0])'*、 *'1+Norm(v)' ...)* の有効なスカラー式を使用することもできます。パラメータにオプションとして与えられる最後の二つのパラメータは、塑性歪み積分に使用されるスキーム (一般化台形則) のパラメータである *'theta* と時間ステップ *dt* です。省略された場合の *theta* のデフォルト値は 1 であり、これは 1 次一貫性のある古典的な後方 Euler 法に対応します。 *theta=1/2* は二次一貫性のある Crank-Nicolson 法 (台形則) に対応します。1/2 から 1 の間の値は有効な値である必要があります。 *dt* のデフォルト値は 'timestep' であり、これは単に (md.set_time_step(dt) により) モデルで定義された時間ステップです。または、任意の式 (データ名、定数値...) を使用できます。時間ステップは、1 つの反復から次の反復に変更できます。 *region* はメッシュ領域です。

使用可能な塑性則は次のとおりです。

- 'Prandtl Reuss' (または 'isotropic perfect plasticity')。硬化のない等方性弾塑性。変数は変位、塑性乗数、塑性歪みです。変位は変数であり、前の時間ステップでの変位 (通常は 'u' と 'Previous_u') に対応する 'Previous_' が先頭に付いた同じ名前の対応するデータを持つ必要があります。塑性乗数には、2 つのバージョン (通常は 'xi' と 'Previous_xi') が必要です。最初のバージョンは、 *unknowns_type* ' が *'DISPLACEMENT_ONLY'* または整数値 *0* の場合はデータとして、 *'unknowns_type* が DISPLACEMENT_AND_PLASTIC_MULTIPLIER または整数値 1 の場合は変数として定義されます。塑性歪みは、mesh_fem または (好ましくは)(*mim* に対応する)im_data に保存されている n x n データテンソルフィールドの形である必要があります。データは、最初の Lame 係数、次の係数 (せん断弾性率)、および一軸降伏応力です。典型的な呼び出し方法は Model.add_small_strain_elastoplasticity_brick(mim, 'Prandtl Reuss', 0, 'u', 'xi', 'Previous_Ep', 'lambda', 'mu', 'sigma_y', '1', 'timestep'); です。重要:この法則は 3 次元表現を実装することに注意してください。2 次元で使用する場合、式は 2 次元に単純に転置されます。平面歪み近似については、以下を参照してください。

- "平面ひずみ Prandtl Reuss" (または "平面ひずみ等方性完全塑性") 前述の法則と同じですが、平面ひずみ近似に適用されます。2 次元でのみ使用できます。

- "Prandtl Reuss 線形硬化" (または "等方性塑性線硬化")。線形等方性および運動硬化を伴う等方性弾塑性。 "Prandtl Reuss" 法と比較される追加の変数、累積塑性歪み。塑性歪みと同様に、時間ステップの終了時にのみ保存されるため、単純なデータが必要です (好ましくは、im_data)。2 つの追加パラメータ: 運動学的硬化係数および等方性。3 次

元表現のみ。典型的な呼び出しは Model.add_small_strain_elastoplasticity_brick(mim, 'Prandtl Reuss linear hardening', 0, 'u', 'xi', 'Previous_Ep', 'Previous_alpha', 'lambda', 'mu', 'sigma_y', 'H_k', H_i', '1', 'timestep'); です。

- "平面ひずみ Prandtl Reuss 線形硬化" (または "平面ひずみ等方性塑性線形硬化")。前の法則と同じですが、平面ひずみ近似に適合しています。2 次元でのみ使用できます。

塑性流動の離散化および実装された塑性則の詳細については GetFEM++ ユーザーマニュアルを参照してください。時間の積分法に関する GetFEM++ ユーザーマニュアル (過渡問題の積分) も参照してください。

重要: *small_strain_elastoplasticity_next_iter* は各時間ステップの終了時、次の時間ステップの前 (と後処理の前: 塑性歪みおよび塑性乗数の値を設定します) に呼び出す必要があります。

add_source_term(*mim*, *expression*, *region=None*)

アセンブリ文字列 *expr* で指定されたソースタームを追加します。これは、領域 *region* で積分法 *mim* を使用してアセンブルされます。残差項のみが考慮されます。式にいくつかの変数が含まれていて、式がポテンシャルである場合、式はすべての変数に関して微分されることに注意してください。 *brickname* はブリックのオプション名です。

add_source_term_brick(*mim*, *varname*, *dataexpr*, *region=None*, **args*)

概要: ind = Model.add_source_term_brick(self, MeshIm mim, string varname, string dataexpr[, int region[, string directdataname]])

モデルに変数 *varname* に比例したソース項を追加します。ソース項は *dataexpr* で表現します。これは高水準汎用アセンブリ言語の任意の正規表現です(ただし複素数モデルの場合は、宣言されたデータでなければりません)。*region*'はオプションで項を追加するメッシュ領域です。追加のオプションデータ *'directdataname* を設定することもできます。対応するデータベクトルは、アセンブリなしで右側に直接追加されます。region が境界である場合、このブリックで Neumann 境界条件を設定することができます。

add_source_term_generic_assembly_brick(*mim*, *expression*, *region=None*)

廃止予定です。 Model.add_source_term() を代わりに使用してください。

add_standard_secondary_domain(*name*, *mim*, *region=-1*)

2 つのドメインの積を統合するために、弱形式言語の表現で使用できるセカンダリドメインをモデルに追加します。 *name* はセカンダリドメインの名前であり、 *mim* はこのドメイン上の積分法であり、 *region* は積分が実行される領域です。

add_theta_method_for_first_order(*varname*, *theta*)

変数 *varname* の時間離散化に theta 法を適用します。変数の最大でも 1 次の時間導関数が存在する場合にのみ有効です。

add_theta_method_for_second_order(*varname*, *theta*)

変数 ‘varname‘の時間離散化に theta 法を適用します。変数の 2 次時間導関数が多くても存在する場合にのみ有効です。

add_twodomain_source_term(*mim*, *expression*, *region*, *secondary_domain*)

Adds a source term given by a weak form language expression like Model.add_source_term() but for an integration on a direct product of two domains, a first specfied by `mim` and `region` and a second one by `secondary_domain` which has to be declared first into the model.

add_variable(*name*, *sizes*)

一定サイズのモデルに変数を追加します。 *sizes* は整数 (スカラー変数またはベクトル変数の場合) かテンソル変数の次元のベクトルです。 *name* は変数名です。

assembly(*option=None*)

すべてのブリックの項を考慮した接線システムのアセンブリです。 *option* を指定する場合、 'build_all' 、 'build_rhs' 、 'build_matrix' とする必要があります。デフォルトでは、接線線形システム全体 (行列と rhs) が構築されます。この関数は、独自のソルバを使用して求解する場合に便利です。

brick_list()

モデルのレンガのリストを出力に出力します。

brick_term_rhs(*ind_brick*, *ind_term=None*, *sym=None*, *ind_iter=None*)

特定の非線形ブリックの項の右側部分にアクセスします。最終的な時間ディスパッチャを考慮しません。まず、右辺のアセンブリを行う必要があります。 *ind_brick* はブリックインデックスです。 *ind_term* はブリック内部の項のインデックスです (デフォルト値は 0)。 *sym* は 2 つの異なる変数に作用する対称項のためにの右辺の 2 番目にアクセスすることです (デフォルトは 0 です)。 *ind_iter* は時間ディスパッチャを使用した場合の反復番号です (デフォルトは 0 です)。

change_penalization_coeff(*ind_brick*, *coeff*)

penalization ブリックを使用して Dirichlet 条件の penalization 係数を変更します。ブリックがこの種類でない場合、この関数の動作は定義されていません。

char()

Model の (ユニークな) 文字列表現を出力します。

これを使用して、2 つの異なる Model オブジェクト間の比較を実行できます。この機能は完成予定です。

clear()

モデルをクリアします。

clear_assembly_assignment()

追加されたアセンブリの割り当てをすべて削除します

compute_Von_Mises_or_Tresca(*varname*, *lawname*, *dataname*, *mf_vm*, *version=None*)

3 次元の非線形弾性のフィールドの Von-Mises 応力または Tresca 応力を *mf_vm* で計算します。 *lawname* は構成則で、'SaintVenant Kirchhoff' 、'Mooney Rivlin' 、'neo Hookean' 、 'Ciarlet Geymonat' だ。 *dataname* は構成則のパラメータのベクトルです。長さは法則によります。これは、定数値の短いベクトル、または可変係数の有限要素法で記述されたベクトルフィールドです。 *version* は 'Von_Mises' または 'Tresca' のいずれかです ('Von_Mises' がデフォルトです)。

compute_elastoplasticity_Von_Mises_or_Tresca(*datasigma*, *mf_vm*, *version=None*)

塑性の場の Von-Mises 応力または Tresca 応力を *mf_vm* で計算し、ベクトル V に戻します。 *datasigma* はメッシュによってサポートされる応力制約値を含むベクトルです。 *version* は 'Von_Mises' または 'Tresca' ('Von_Mises' がデフォルト) でなければなりません。

compute_finite_strain_elasticity_Von_Mises(*lawname*, *varname*, *params*, *mf_vm*, *region=None*)

3 次元の非線形弾性のフィールド *varname* の Von-Mises 応力を *mf_vm* で計算します。 *lawname* は構成則であり、有効な名前である必要があります。 *params* はパラメータ則です。定数値の短いベクトルにすることも、モデルのデータや変数に依存することもできます。高水準汎用アセンブリを使用します。

compute_finite_strain_elastoplasticity_Von_Mises(*mim*, *mf_vm*, *lawname*, *unknowns_type*, *varnames=None*, **args*)

概要: V = Model.compute_finite_strain_elastoplasticity_Von_Mises(self, MeshIm mim, MeshFem mf_vm, string lawname, string unknowns_type, [, string varnames, ...] [, string params, ...] [, int region = -1])

塑性場の Von-Mises または Tresca 応力を *mf_vm* で計算し、ベクトル V に返します。最初の入力パラメータは関数 'finite strain elastoplasticity next iter' と同様です。

compute_isotropic_linearized_Von_Mises_or_Tresca(*varname*, *dataname_lambda*, *dataname_mu*, *mf_vm*, *version=None*)

Von-Mises 応力またはフィールドの Tresca 応力 (3 次元の等方性線形化弾性にのみ有効) を計算します。 *version* は 'Von_Mises' または 'Tresca' ('Von_Mises' がデフォルト) でなければなりません。Lame 係数によってパラメータ化されます。

compute_isotropic_linearized_Von_Mises_pstrain(*varname*, *data_E*, *data_nu*, *mf_vm*)

平面ひずみを仮定した 3 次元または 2 次元における等方性線形化弾性の変位場の Von-Mises 応力を計算します。Young 率と Poisson 比によってパラメータ化されます。

compute_isotropic_linearized_Von_Mises_pstress(*varname*, *data_E*, *data_nu*, *mf_vm*)

平面応力を仮定した 3 次元または 2 次元における等方性線形化弾性の変位場の Von-Mises 応力を計算します。Young 率と Poisson 比によってパラメータ化されます。

compute_plastic_part(*mim*, *mf_pl*, *varname*, *previous_dep_name*, *projname*, *data-lambda*, *datamu*, *datathreshold*, *datasigma*)

プラスチック成形品を *mf_pl* で計算し、ベクトル V に返します。 *datasigma* はメッシュでサポートされる応力拘束値を含むベクトルです。

compute_second_Piola_Kirchhoff_tensor(*varname*, *lawname*, *dataname*, *mf_sigma*)

3 次元非線形弾性の場の第 2Piola Kirchhoff 応力テンソルの mf_sigma を計算します。 *lawname* は構成則で 'SaintVenant Kirchhoff' 、 'Mooney Rivlin' 、 'neo Hookean' 、あるいは 'Ciarlet Geymonat' となります。 *dataname* は構成則のパラメータのベクトルです。長さは法則によります。これは、定数値の短いベクトル、または可変係数の有限要素法で記述されたベクトルフィールドです。

contact_brick_set_BN(*indbrick*, *BN*)

基本的な接触/摩擦ブリックの BN マトリックスの設定に使用できます。

contact_brick_set_BT(*indbrick*, *BT*)

摩擦ブリックとの基本接触の BT マトリックスの設定に使用できます。

define_variable_group(*name*, *varname=None*, **args*)

概要: Model.define_variable_group(self, string name[, string varname, ...])

補間 (主にレイトレーシング補間変換の変数のグループを定義します。

del_macro(*name*)

汎用アセンブリ言語用に定義済みのマクロを削除します。

delete_brick(*ind_brick*)

モデルから変数またはデータを削除します。

delete_variable(*name*)

モデルから変数またはデータを削除します。

disable_bricks(*bricks_indices*)

ブリックを無効にします (ブリックは接線線形システムの構築に関与しなくなります)。

disable_variable(*varname*)

求解のため変数 (とその添付乗数) を使用不可にします。次の求解は、残りの変数に対してのみ実行されます。これにより、モデルの異なる部分を別々に求解できます。変数の強連成がある場合は、固定小数点方式を使用できます。

displacement_group_name_of_Nitsche_large_sliding_contact_brick(*indbrick*)
既存の大きな滑り接触ブリックの滑りデータに対応する変数のグループの名前を指定します。

displacement_group_name_of_large_sliding_contact_brick(*indbrick*)
既存の大きな滑り接触ブリックの滑りデータに対応する変数のグループの名前を指定します。

display()
Model オブジェクトの概要が表示されます。

elastoplasticity_next_iter(*mim*, *varname*, *previous_dep_name*, *projname*, *datalambda*, *datamu*, *datathreshold*, *datasigma*)
古い (廃止された) 弾塑性ブリックとともに使用して、イテレーションから次のイテレーションに渡します。次のイテレーションの応力拘束 sigma を計算して保存します。 'mim' は、計算に使用する積分法です。 'varname' がこの問題の主要な変数です。 'previous_dep_name' は前の時間ステップでの変位を表します。 'projname' は使用する投影のタイプです。現時点では 'Von Mises' または 'VM' でなければなりません。 'datalambda' および 'datamu' は、材料の Lame 係数です。 'datasigma' は新しい応力拘束値を格納するベクトルです。

enable_bricks(*bricks_indices*)
無効なブリックを有効にします。

enable_variable(*varname*)
使用不可の変数を使用可能にします (付属の乗数)。

finite_strain_elastoplasticity_next_iter(*mim*, *lawname*, *unknowns_type*, *varnames=None*, **args*)
概要: Model.finite_strain_elastoplasticity_next_iter(self, MeshIm mim, string lawname, string unknowns_type, [, string varnames, ...] [, string params, ...] [, int region = -1])

有限ひずみ塑性ブリックの時間ステップから別のステップへの移行を可能にする関数です。パラメータは、 *add_finite_strain_elastoplasticity_brick* とまったく同じでなければなりません。説明については、この関数のドキュメントを参照してください。基本的に、このブリックは塑性歪みと塑性乗数を計算し、次のステップのために保存します。現在実装されている唯一の Simo-Miehe の法則に対して、この関数は *varnames* の最後の 2 つのエントリで定義されている状態変数を更新し、 *varnames* の 2 番目のエントリとして与えられる塑性乗数フィールドをリセットします。

first_iter()
時間積分スキームの最初の反復の前に実行します。

from_variables()
モデルの変数を連結したモデルのすべての自由度 (独自のソルバを使用して求解するのに便利) のベクトルを返します。

get_time()
現在時刻に対応するデータ *t* の値を指定します。

get_time_step()
タイムステップの値を指定します。

interpolation(*expr*, **args*)
概要: V = Model.interpolation(self, string expr, {MeshFem mf | MeshImd mimd | vec pts, Mesh m}[, int region[, int extrapolation[, int rg_source]]])

mesh_fem *mf* または mesh_im_data *mimd* またはメッシュ *m* 上の点の集合 *pts* を基準にして特定の式を補間します。式は、モデルの変数とデータへの参照を含む可能性があります、高水準汎用アセンブリ言語に従って有効である必要があります。

オプション **外挿** と *rg_source* は、点 *pts* の集合に関する補間のためのものです。

interval_of_variable(*varname*)
モデルの線形システムの変数 *varname* の間隔を指定します。

is_complex()
モデルが実数の場合は 0 を返し、複素数の場合は 1 を返します。

list_residuals()
モデルに含まれるすべての項に対応する残差を出力します。

local_projection(*mim*, *expr*, *mf*, *region=None*)
mesh_fem *mf* を基準にして、式の要素ごとの L2 投影を作成します。この mesh_fem は不連続である必要があります。式は、モデルの変数とデータへの参照を含む可能性があります、上位レベルの汎用アセンブリ言語に従い有効である必要があります。

matrix_term(*ind_brick*, *ind_term*)
存在する場合、ブリック ind_brick のマトリックス項 ind_term を返します。

memsize()
モデルが使用するメモリ量 (バイト単位) の概算値を返します。

mesh_fem_of_variable(*name*)
変数またはデータの *mesh_fem* へのアクセスを許可します。

mult_varname_Dirichlet(*ind_brick*)

Dirichlet ブリックの乗数変数の名前を指定します。ブリックが乗数ブリックを持つ Dirichlet 条件ではない場合、この関数の動作は未定義です。

nbdof()

モデルの自由度の総数を返します。

next_iter()

時間積分スキームの各反復の終了時に実行されます。

perform_init_time_derivative(*ddt*)

この関数を呼び出すことにより、時間積分スキームによって必要とされる導関数に対応するデータを初期化するためには、次の求解が (非常に) 小さな時間ステップ (主に時間問題では 1 次の初期時間微分、時間問題では 2 次の 2 次時間微分) の解を計算することを示します。次の求解では、変数の値は変更されません。

resize_variable(*name*, *sizes*)

モデルの一定サイズ変数のサイズを変更します。 *sizes* は (スカラー変数またはベクトル変数の場合) 整数かテンソル変数の次元のベクトルです。 *name* は変数名です。

rhs()

接線問題の右辺を返します。

set_element_extrapolation_correspondence(*transname*, *elt_corr*)

要素外挿補間変換の対応マップを変更します。

set_private_matrix(*indbrick*, *B*)

内部に疎行列がある特定のブリック (陽なレンガ: ' 拘束ブリック' と ' 陽な行列ブリック') では、この行列を設定します。

set_private_rhs(*indbrick*, *B*)

内部右辺ベクトル (陽なブリック: ' 拘束ブリック' と ' 陽なブリック') を持つ特定のレンガでは、この rhs を設定します。

set_time(*t*)

現在時刻 *t* に対応するデータの値を *t* に設定します。

set_time_step(*dt*)

時間ステップの値を *dt* に設定します。この値は、すべてのワンステップスキームでステップから別のステップに変更できます (すなわち現時点では、提案されているすべての時間積分法)。

set_variable(*name*, *V*)

変数またはデータの値を設定します。 *name* はデータ名です。

shift_variables_for_time_integration()

モデルの変数を、時間積分スキームの前の時間ステップの他の値に対応するデータにシフトするために使用される関数です。時間積分スキームが宣言されている変数ごとに、スキームが呼び出されてシフトが実行されます。この関数は、2 つのタイムステップの間に呼び出す必要があります。

sliding_data_group_name_of_Nitsche_large_sliding_contact_brick(*indbrick*)

既存の大きな滑り接触ブリックの滑りデータに対応する変数のグループの名前を指定します。

sliding_data_group_name_of_large_sliding_contact_brick(*indbrick*)

既存の大きな滑り接触ブリックの滑りデータに対応する変数のグループの名前を指定します。

small_strain_elastoplasticity_Von_Mises(*mim*, *mf_vm*, *lawname*, *unknowns_type*, *varnames=None*, **args*)

概要: V = Model.small_strain_elastoplasticity_Von_Mises(self, MeshIm mim, MeshFem mf_vm, string lawname, string unknowns_type [, string varnames, ...] [, string params, ...] [, string theta = '1' [, string dt = 'timestep']] [, int region])

この関数は、 *mf_vm* で近似された小さな歪みの弾塑性項に関するフォンミーゼス応力場を計算し、その結果を *VM* に格納します。その他のパラメータはすべて、*add_small_strain_elastoplasticity_brick* とまったく同じでなければなりません。この関数を呼び出す前に、 *small_strain_elastoplasticity_next_iter* を呼び出す必要があります。

small_strain_elastoplasticity_next_iter(*mim*, *lawname*, *unknowns_type*, *varnames=None*, **args*)

概要: Model.small_strain_elastoplasticity_next_iter(self, MeshIm mim, string lawname, string unknowns_type [, string varnames, ...] [, string params, ...] [, string theta = '1' [, string dt = 'timestep']] [, int region = -1])

微小ひずみ塑性ブリックの時間ステップ間を通過できる関数。パラメータは、*add_small_strain_elastoplasticity_brick* とまったく同じでなければなりません。説明については、この関数のドキュメントを参照してください。基本的に、このブリックは塑性歪みと塑性乗数を計算し、次のステップのために保存します。さらに、計算されたディスプレイスメントが、前のタイムステップのディスプレイスメントを格納するデータにコピーされます (通常は 'u' から 'Previous_u')。これは、 *compute_small_strain_elastoplasticity_Von_Mises* を使用する前に呼び出す必要があります。

solve(**args*)

概要: (nbit, converged) = Model.solve(self[, ...])

標準の getfem ソルバーを実行します。

必要に応じて、独自のソルバを使用できるようにする必要があります (Model.tangent_matrix() などで接線行列とその右辺を得ることができます)。

さまざまなオプションを指定できます。

- 'noisy' または 'very_noisy'　ソルバが進行状況を示す情報が表示します (残差値など)。
- 'max_iter', int NIT　最大反復回数を設定します。
- 'max_res', @float RES　目標残差値を設定します。
- 'diverged_res', @float RES　反復法が発散したと見なされる残差のしきい値を設定します (デフォルトは 1e200 です)。
- 'lsolver', string SOLVER_NAME　線形システム (デフォルト値は 'auto' で、getfem が自分で選択できるようになっています) に使用するソルバーを明示的に選択します。指定できる値は 'superlu' 、(サポートされている場合) 'mumps' 、 'cg/ildlt' 、 'gmres/ilut' 、および 'gmres/ilut' です。
- 'lsearch', string LINE_SEARCH_NAME　線形システムで使用される線検索方法を明示的に選択します (デフォルト値は 'default' です)。指定できる値は、'simplest' 、 'systematic' 、 'quadratic' 、または 'basic' です。

反復法が使用される場合、反復回数を返します。

変数のサブセット (無効にされた変数はデータとみなされます) に関してのみ問題を求解するために、いくつかの変数 (Model.disable_variable() を参照してください) を無効にすることが可能であることに注意してください。例えば、全体 Newton 法を固定小数点法で置き換えることができます。

tangent_matrix()

モデルに格納されている接線行列を返します。

test_tangent_matrix(*EPS=None*, **args*)

概要: Model.test_tangent_matrix(self[, scalar EPS[, int NB[, scalar scale]]])

ランダムな位置と方向 (新しく作成されたブリックをテストするのに便利) で、接線行列の整合性をテストします。EPS は微分の差分計算のための小パラメータの値であり、ランダム方向 (デフォルトは 1E-6) です。 *NN* はテストの数です (デフォルトは 100 です)。 *scale* は現在位置の周囲のランダムな位置 (デフォルトは 1 で、0 が許容値です) のパラメータです。ランダム位置の各自由度は [current-scale, current+scale] の範囲内で選択されます。

test_tangent_matrix_term(*varname1*, *varname2*, *EPS=None*, **args*)

概要: Model.test_tangent_matrix_term(self, string varname1, string varname2[, scalar EPS[, int NB[, scalar scale]]])

ランダムな位置と方向 (新しく作成されたレンガをテストするのに便利) で、接線行列の一部の整合性をテストします。インクリメントは、変数 *varname2* に対してのみ行われ、 *varname1*'に対応する残差の部分でテストされます。これは、接線マトリックスの項 *('varname1*, *varname2*) のみがテストされることを意味します。EPS は微分の差分計算のための微小パラメータの値 (デフォルトは 1E-6) であり、ランダム方向です。 *NN* はテストの数です (デフォルトは 100 です)。 *scale* は現在位置の周囲のランダムな位置 (デフォルトは 1 で、0 が許容値です) のパラメータである。ランダム位置の各自由度は [current-scale, current+scale] の範囲内で選択されます。

to_variables(*V*)
ベクトル *V* でモデルの変数の値を設定します。通常、ベクトル *V* は接線線形システム (独自のソルバを使用して問題を解決するのに便利) のソルバの結果です。

transformation_name_of_Nitsche_large_sliding_contact_brick(*indbrick*)
既存の大きな滑り接触ブリックの滑りデータに対応する変数のグループの名前を指定します。

transformation_name_of_large_sliding_contact_brick(*indbrick*)
既存の大きな滑り接触ブリックの滑りデータに対応する変数のグループの名前を指定します。

variable(*name*)
変数またはデータの値を指定します。

variable_list()
モデルの変数と定数のリストを出力に出力します。

7.16 Precond

class Precond(**args*)
GeFEM Precond オブジェクト

前処理行列は、REAL 値または COMPLEX 値を記憶することができます。getfem 疎行列と Matlab 疎行列が使用可能です。

Precond オブジェクトの汎用的なコンストラクタ

- `PC = Precond('identity')` 実数の前処理行列を作成します。
- `PC = Precond('cidentity')` COMPLEX 単位前処理行列を作成します。
- `PC = Precond('diagonal', vec D)` 対角前処理行列を作成します。

- `PC = Precond('ildlt', SpMat m)` (対称) 疎行列 *m* の ILDLT(Cholesky) 前処理行列を作成します。この前処理行列は (fill-in 無しの) *m* と同じ非ゼロパターンです。
- `PC = Precond('ilu', SpMat m)` 疎行列 *m* に対する ILU(不完全な LU) 前処理行列を作成します。この前処理行列は、(fill-in 無し) *m* と同じじ非ゼロパターンを有します。
- `PC = Precond('ildltt', SpMat m[, int fillin[, scalar threshold]])` (対称の) 疎行列 *m* の ILDLTT(密な Cholesky) 前処理行列を作成します。前処理行列は、各行上に追加の非ゼロのエントリを最大 *fillin* 個追加することができます。 *fillin* のデフォルト値は 10 で、デフォルトの閾値は 1e-7 です。
- `PC = Precond('ilut', SpMat m[, int fillin[, scalar threshold]])` 疎行列 *m* に対する ILUT(フィルされている不完全な LU) 前処理行列を作成します。前処理行列は、各行上に追加の非ゼロのエントリを多くとも *fillin* 個追加することができます。 'fillin'のデフォルト値は 10 で、デフォルトの閾値は 1e-7 です。
- `PC = Precond('superlu', SpMat m)` SuperLU を使用して、疎行列 *m* の正確な因数分解を構築します。この前処理行列は、SuperLU をサポートする getfem-interface が構築されている場合にのみ使用できます。LU 分解は、3 次元問題の場合、メモリをすべて消費する可能性が高いことに注意してください。
- `PC = Precond('spmat', SpMat m)` 疎行列によって明示的に与えられる前処理行列です。

char()
Precond の (一意な) 文字列表現を出力します。

これを使用して、2 つの異なる Precond オブジェクト間の比較を実行できます。この機能は完成予定です。

display()
Precond オブジェクトの概要が表示されます。

is_complex()
前処理行列が複素数値を格納する場合は 1 を返します。

mult(*V*)
与えられたベクトルに前処理行列を適用します。

size()
前処理行列の次元を返します。

tmult(*V*)
転置された前処理行列を与えられたベクトルに適用します。

type()
前処理の種類を 表す文字列 ('ilu', 'ildlt',..) を返します。

7.17 Slice

class Slice(**args*)
GeFEM Slice オブジェクト

メッシュスライスの作成をします。メッシュスライスは、補間が非常に高速な P1 不連続メッシュ分割に非常によく似ています。スライスは、メッシュオブジェクトとスライス操作の記述により作成されます。次に例を示します。

```
sl = Slice(('planar',+1,[[0],[0]],[[0],[1]]), m, 5)
```

は元のメッシュを半空間 {y>0} で切り取ります。元のメッシュ m の各凸包を単純化します (例えば、四角形は 2 つの三角形に分割されます)。そして各単体は 5 回リファインされます。

スライス操作には次のものがあります。

- 平面、球または円柱を使った切断
- スライスの交差または結合
- 等値面/体
- "points", "streamlines" (下記参照)

最初の引数が Mesh ではなく MeshFem *mf* で、その後に *mf* フィールド *u* が続く場合、変形 *u* はスライス操作の前にメッシュに適用されます。

最初の引数はスライスにすることもできます。

Slice オブジェクトの汎用的なコンストラクタです。

- `sl = Slice(sliceop, {Slice sl|{Mesh m| MeshFem mf, vec U}, int refine}[, mat CVfids])` *sliceop* 操作でスライスを作成します。

  sliceop 操作は Tuple か List で指定されます。追加の括弧を忘れないでください! 最初の要素は操作の名前で、その後にスライスオプションが続きます。

  – ('none') : メッシュをカットしません。

  – ('planar', int orient, vec p, vec n) : 平面カット。 *p* と *n* は半空間を定義し、 *p* は半空間の境界に属する点であり、 *n* はその法線です。 *orient* が -1 (または 0, +1) に等しい場合、スライス操作は半空間の "内側" (または "境界", "外側") でメッシュを切断しま

す。 *orient* を +2 に設定すると、メッシュはスライスされますが、外側と内側の両方の部分は保持されます。

- ('ball', int orient, vec c, scalar r) : 中心 *c* と半径 *r* の球でカットします。
- ('cylinder', int orient, vec p1, vec p2, scalar r) : 軸が *(p1, p2)* 線で半径が *r* の円柱で切断します。
- ('isovalues', int orient, MeshFem mf, vec U, scalar s) : (MeshFem *mf* で定義された) フィールド *U* の等値面を使用して切断します。結果は *orient* の値に応じて *{x such that :math:'U(x) leq s}'* または *{x such that 'U'(x)='s}'* または *{x such that 'U'(x) >= 's}'* になります。
- ('boundary'[, SLICEOP]) : SLICEOP の結果の境界を返します。ここで、SLICEOP はスライシング操作です。SLICEOP が指定されていない場合は、メッシュ全体 (つまり、 ('boundary',{'none'})) と見なされます。
- ('explode', mat Coef) : メッシュの 'exploded' 表示を構築します。各凸包が $(0 < \text{Coef} \leq 1)$ 収縮されます。3 次元凸包の場合、面のみが保持されます。
- ('union', SLICEOP1, SLICEOP2) : スライス操作の和集合を返します。
- ('intersection', SLICEOP1, SLICEOP2) : スライスの交差操作を返します。例えば

```
sl = Slice(('intersection',('planar',+1,[[0],[0],[0]],[[0],[0],
↪[1]]),
                           ('isovalues',-1,mf2,u2,0)),mf,u,5)
```

- ('comp', SLICEOP) : スライス操作の補完を返します。
- ('diff', SLICEOP1, SLICEOP2) : スライス操作の差を返します。
- ('mesh', Mesh m) : スライスされたメッシュと別のメッシュの交差部分のスライスを作成します。このスライスは、すべての単項が各メッシュの凸包に厳密に含まれるものです。

- `sl = Slice('streamlines', MeshFem mf, mat U, mat S)` *S* の列で与えられるシード点を使って、(ベクトル) フィールド *U* の流線を計算します。
- `sl = Slice('points', Mesh m, mat Pts)` Pts のカラムで指定された点で構成される "スライス" を返します。(指定されたスパース点の集合上の補間に役立ちます。`gf_compute('interpolate on',sl)` を参照してください。
- `sl = Slice('load', string filename[, Mesh m])` テキストファイルからスライス (および、引数として指定されていない場合、リンクされたメッシュ) を読み込みます。

area()
スライスの面積を返します。

char()
スライスの (一意な) 文字列表現を出力します。

これを使用して、2 つの異なる Slice オブジェクト 間の比較を実行できます。この機能は完成予定です。

cvs()
スライスに含まれる元のメッシュの凸包のリストを返します。

dim()
スライスの次数 (2 次元メッシュであれば 2 など) を返します。

display()
Slice オブジェクト の概要が表示されます。

edges()
スライスに含まれるリンクメッシュのエッジを返します。

P はすべてのエッジ頂点のリストを含み、 *E1* は *P* における各メッシュエッジのインデックスを含み、 *E2* はスライスの境界上にある各 "エッジ" のインデックスを含みます。この機能は、ポスト 処理の目的以外には使用できません。

export_to_dx(*filename*, **args*)
概要: Slice.export_to_dx(self, string filename, ...)

スライスを OpenDX に書き出します。

filename の後には、以下のオプションを使うことができます。

- 'ascii' を使用しない場合、ファイルには (ポータブルではないが高速な) バイナリデータが格納されます。
- 'edges' を使用すると、スライスではなくオリジナルのメッシュのエッジが書き込まれます。
- 'append' を使用すると、opendx ファイルは上書きされず、新しいデータがファイルの最後に追加されます。

複数のデータセットを書き込むこともできますが、それらを一覧表示するだけです。各データセットは次のいずれかで構成されます。

- スライス上で補間されるフィールド (スカラー、ベクトル、テンソル) で、その後にオプションの名前が続きます。
- mesh_fem とフィールドを指定し、その後にオプションの名前を指定します。

export_to_pos (*filename*, *name=None*, **args*)

概要: Slice.export_to_pos(self, string filename[, string name][[,MeshFem mf1], mat U1, string nameU1[[,MeshFem mf1], mat U2, string nameU2,...])

スライスを Gmsh にエクスポートします。

複数のデータセットを書き込むこともできますが、それらを一覧表示するだけです。各データセットは次のいずれかで構成されます。

- スライス上で補間されるフィールド (スカラー、ベクトル、テンソル)。
- mesh_fem とフィールド。

export_to_pov (*filename*)

スライスの三角形を POV-RAY に書き出します。

export_to_vtk (*filename*, **args*)

概要: Slice.export_to_vtk(self, string filename, ...)

スライスを VTK にエクスポートします。

filename の後には、以下のオプションを使うことができます。

- 'ascii' を使用しない場合、ファイルには (ポータブルではないが高速な) バイナリデータが格納されます。
- 'edges' を使用すると、スライスではなくオリジナルのメッシュのエッジが書き込まれます。

複数のデータセットを書き込むこともできますが、それらを一覧表示するだけです。各データセットは次のいずれかで構成されます。

- スライス上で補間されるフィールド (スカラー、ベクトル、テンソル) で、その後にオプションの名前が続きます。
- mesh_fem とフィールドを指定し、その後にオプションの名前を指定します。

例

- Slice.export_to_vtk('test.vtk', Usl, 'first_dataset', mf, U2, 'second_dataset')
- Slice.export_to_vtk('test.vtk', 'ascii', mf,U2)
- Slice.export_to_vtk('test.vtk', 'edges', 'ascii', Uslice)

interpolate_convex_data (*Ucv*)

メッシュの各凸包に指定されたデータをスライスの節点で補間します。

入力配列 *Ucv* は任意の数の次元を持つことができますが、その最後の次元は Mesh.max_cvid() に等しくなければなりません。

使用例: Slice.interpolate_convex_data(Mesh.quality()) 。

linked_mesh()
スライスが作成されたメッシュを返します。

memsize()
Slice オブジェクトが使用するメモリーの量 (バイト単位) を返します。

mesh()
スライスが作成されたメッシュを返します ('リンクメッシュ' と同一)。

nbpts()
スライス内のポイントの数を返します。

nbsplxs(*dim=None*)
スライス内の単体の数を返します。

スライスには、ポイント (dim0 の単体)、セグメント (次元 1 の単体)、三角形などが含まれる場合があるため、オプションの引数 *dim* が使用されている場合を除いて、結果はサイズ Slice.dim()+1 のベクトルになります 。

pts()
点座標のリストを返します。

set_pts(*P*)
スライスの点を置き換えます。

新しいポイント *P* は行列の列に格納されます。この関数を使用して、スライスに変形を適用したり、スライスの次元を変更したりできます (*P* の行数は Slice.dim() と同じである必要はありません)。

splxs(*dim*)
次元 *dim* の単体のリストを返します。

出力では、S には 'dim+1' 行があり、各列には単体のポイント番号が含まれています。ベクトル *CV2S* を使用して、スライス内に格納されている任意の凸包の単体のリストを見つけることができます。たとえば、'S[:,CV2S[4]:CV2S[5]]' とすると、第 4 の凸包の単体のリストが得られます。

7.18 Spmat

class Spmat(**args*)

GeFEM Spmat オブジェクト

getfem++ 形式で新しい疎行列を作成します。これらの疎行列は、Matlab で使用される形式である CSC(圧縮された疎の列) として保存することも、WSC(getfem の内部形式) として保存することもできます。CSC 行列は書き込み可能ではありませんが (これは非常に非効率的です)、ベクトルとの乗算とメモリ使用量のために最適化されています。WSC は書き込み可能であり、ランダムな読み取り/書き込み操作に関して非常に高速です。しかし、メモリオーバヘッドは CSC 行列よりも高く、行列ベクトル乗算では若干遅いです。

デフォルトでは、新しく作成されたマトリックスはすべて WSC マトリックスとして構築されます。これは後で `Spmat.to_csc(...)` か、あるいは getfem(例えば `gf_linsolve()` は行列を CSC に変換します) によって自動的に変更できます。

マトリックスには実数値または複素数値を格納できます。

Spmat オブジェクトの汎用的なコンストラクタ

- `SM = Spmat('empty', int m [, int n])` 新しい空の (すなわち、ゼロで満たされた) 次元 *m x n* 疎行列を作成します。 *n* を省略すると、行列の次元は *m x m* になります。
- `SM = Spmat('copy', mat K [, list I [, list J]])` (SpMat かもしれない) マトリックス K を複製します。インデックス *I* 及び/又は *J* が与えられる場合、行列は *K* の部分行列となります。例えば

```
m = Spmat('copy', Spmat('empty',50,50), range(40), [6, 7, 8, 3, 10])
```

  は 40x5 の行列を返します。

- `SM = Spmat('identity', int n)` *n x n* 次の単位行列を作成します。
- `SM = Spmat('mult', Spmat A, Spmat B)` 疎行列 A と B の積の疎行列を作成します。 *A* と *B* が両方とも実数であるか、両方とも複素数である必要があります。`Spmat.to_complex()` を使う必要があるかもしれません。
- `SM = Spmat('add', Spmat A, Spmat B)` 疎行列 A と B の和の疎行列を作成します。複素数行列を持つ実数行列の追加が可能です。
- `SM = Spmat('diag', mat D [, ivec E [, int n [,int m]]])` 対角行列を作成します。 *E* が与えられる場合、 *D* は行列であり、 *E* の各列は *D* の対応する列の部分対角番号を含みます。
- `SM = Spmat('load','hb'|'harwell-boeing'|'mm'|'matrix-market',`

`string filename`) Harwell-Boeing や Matrix-Market のファイルから疎行列を読みます。

add(*I*, *J*, *V*)
小行列 'M(I,J)' に *V* を加えます。

V は疎行列でも密行列でもかまいません。

assign(*I*, *J*, *V*)
V を小行列 'M(I,J)' にコピーします。

V は疎行列でも密行列でもかまいません。

char()
Spmat の (ユニークな) 文字列表現を出力します。

これを使用して、2 つの異なる Spmat オブジェクト 間の比較を実行できます。この機能は完成予定です。

clear(*I=None*, **args*)
概要: Spmat.clear(self[, list I[, list J]])

行列のゼロ以外のエントリを削除します。

行列全体ではなく、小行列をクリアするには、オプションの引数 *I* と *J* を指定します。

conjugate()
行列の各要素を共役させます。

csc_ind()
CSC ストレージの 2 つの通常のインデックス配列を返します。

M が CSC 行列として保存されていない場合、それは CSC に変換されます。

csc_val()
M の 0 以外のすべての要素の値の配列を返します。

M が CSC 行列として保存されていない場合、それは CSC に変換されます。

determinant()
MUMPS を使用して計算された行列式を返します。

diag(*E=None*)
M の対角をベクトルとして返します。

E が使用されている場合、rank が E で与えられている部分対角を返します。

dirichlet_nullspace(*R*)
Dirichlet 条件 *M.U=R* を解きます。

最小 L2 ノルムを持つ解 *U0* が (アセンブリング) 制約行列 M(すなわち、偏微分線形方程式系はこの部分空間上で解かれるべきである) のカーネルの直交基底を含む疎行列 N とともに返されます。

M.U=R の制約がある *K.U = B*

は次のように変わります

(N'.K.N).UU = N'.B ただし *U = N.UU + U0*

display()
Spmat オブジェクトの簡単な概要が表示されます。

full(*I=None*, **args*)
概要: Sm = Spmat.full(self[, list I[, list J]])

(小) 行列全体を返します。

オプションの引数 *I* および *J* は、抽出される行および列のサブインターバルです。

is_complex()
行列に複素数が含まれている場合、1 を返します。

mult(*V*)
ベクトル *V* と疎行列 *M* の積。

行列と行列の乗算については、 Spmat('mult') を参照してください。

nnz()
疎行列に格納されている非 NULL 値の数を返します。

save(*format*, *filename*)
疎行列をエクスポートします。

ファイルの形式は、Harwell-Boeing の場合は 'hb' 、Matrix-Market の場合は 'mm' となります。

scale(*v*)
行列をスカラー値 *v* で乗算します。

set_diag(*D*, *E=None*)
マトリックスの対角 (または副対角) を変更します。

E が与えられる場合、 *D* は行列であり 、 *E* の各列は *D* の対応する列の部分対角の番号を含みます。

size()
ni と *nj* が行列の次元であるベクトルを返します。

storage()
行列で現在使用されているストレージの種類を返します。

ストレージは文字列 'CSC' または 'WSC' として返されます。

tmult(*V*)
ベクトル *V* と転置された *M* は (*M* が複素数の場合は共役) の積。

to_complex()
複素数を格納します。

to_csc()
マトリックスを CSC ストレージに変換します。

行列-ベクトル乗算には CSC ストレージが推奨されます。

to_wsc()
行列を WSC ストレージに変換します。

WSC ストレージでは、読み取りと書き込み処理が非常に高速です。

transconj()
行列を転置して共役させます。

transpose()
行列を転置します。

7.19 モジュール asm

asm_generic(*mim*, *order*, *expression*, *region*, *model=None*, **args*)
概要: (...) = asm_generic(MeshIm mim, int order, string expression, int region, [Model model, ['Secondary_domain', 'name',]] [string varname, int is_variable[, {MeshFem mf, MeshImd mimd}], value], ['select_output', 'varname1'[, 'varname2]], ...)

ボリュームアセンブリまたは境界アセンブリの高水準の汎用アセンブリ手順。

積分法 *mim* を使用して、インデックス *region* (-1 はメッシュのすべての要素を意味します) のメッシュ領域上で *expression* の汎用アセンブリを実行します。同じメッシュを積分法と、変数に対応するすべての有限要素法または mesh_im_data で共有する必要があります。

order は、(スカラー) ポテンシャル (order=0) 、(ベクトル) 残差 (order=1)、または接線 (行列)(order=2) のいずれかが計算されることを示します。

model はモデルのすべての変数とデータを考慮に入れることができるオプションのパラメータです。モデルのすべての使用可能な変数は、たとえ *expression* に存在しなくても、全体系での

自由度に対応する返されたベクトル/行列の空間を占めることに注意してください。

変数と定数 (データ) は、領域番号 (またはオプションでモデル) の後にリストされます。変数/定数ごとに、最初に名前を指定する必要があります (アセンブリ文字列で参照されるとおり)。次に、変数または定数を宣言するために、それぞれ 1 または 0 に等しい整数が必要です。次に、fem 変数/定数の場合は有限要素法、積分点で定義されたデータの場合は mesh_im_data、および変数/定数の値を表すベクトルが必要です。任意の数の変数/定数を指定できます。変数と定数の違いは、試験関数は変数に対してのみ使用でき、定数に対しては使用できないことです。

select_output は出力ベクトル (*order* が 1 の場合) または行列 (*order* が 2 の場合) を指定された変数の自由度まで減らすことができるオプションのパラメータです。ベクトル出力には 1 つの変数を指定し、行列出力には 2 つの変数を指定する必要があります。

複数の変数が与えられた場合、正接行列/残差ベクトルのアセンブリは、関数の呼び出しの順序を考慮して行われます (最初の変数の自由度、次に 2 番目の変数の自由度、というように続きます)。モデルが提供されている場合、一部のモデル変数が *expression* に現れない場合でも、モデルのすべての自由度が最初にカウントされます。

例えば、ベクトルフィールド "u" の L2 ノルムは

```
gf_compute('L2 norm') or with the square root of:

gf_asm('generic', mim, 0, 'u.u', -1, 'u', 1, mf, U);
```

スカラー場の不均質 Laplacian 剛性行列は以下で評価できます。

```
gf_asm('laplacian', mim, mf, mf_data, A) or equivalently with:

gf_asm('generic', mim, 2, 'A*Grad_Test2_u.Grad_Test_u', -1, 'u', 1, mf,
 U, 'A', 0, mf_data, A);
```

asm_mass_matrix (*mim*, *mf1*, *mf2=None*, **args*)

概要: M = asm_mass_matrix(MeshIm mim, MeshFem mf1[, MeshFem mf2[, int region]])

質量行列のアセンブリ。

SpMat オブジェクトを返します。

asm_laplacian (*mim*, *mf_u*, *mf_d*, *a*, *region=None*)

Laplacian 問題の行列の組み立て。

$\nabla \cdot (a(x)\nabla u)$ ここで a はスカラーです。

SpMat オブジェクトを返します。

asm_linear_elasticity(*mim*, *mf_u*, *mf_d*, *lambda_d*, *mu_d*, *region=None*)

線形 (等方性) 弾性問題のためのマトリックスの集合。

$\nabla \cdot (C(x) : \nabla u)$ ここで C は *lambda_d* と *mu_d* によって定義されます。

SpMat オブジェクトを返します。

asm_nonlinear_elasticity(*mim*, *mf_u*, *U*, *law*, *mf_d*, *params*, **args*)

概要: TRHS = asm_nonlinear_elasticity(MeshIm mim, MeshFem mf_u, vec U, string law, MeshFem mf_d, mat params, {'tangent matrix'|'rhs'|'incompressible tangent matrix', MeshFem mf_p, vec P|'incompressible rhs', MeshFem mf_p, vec P})

非線形弾性の項 (接線行列と右辺) をアセンブルします。

現在の時間ステップでは解 *U* が必要です。 *law* は次の中から選択することができます。

- 'SaintVenant Kirchhoff': 線形は避けるべきです)。この法則には、パラメータとして、lambda および mu と呼ばれる 2 つの通常の Lame 係数があります。
- 'Mooney Rivlin': この法則には C1、C2、D1 という 3 つのパラメータがあります。前に 'compressible' または 'incompressible' を付けることで、特定のバージョンを強制するころができます。デフォルトでは、最初の 2 つの材料係数のみを必要とする非圧縮性バージョンが考慮されます。
- 'neo Hookean': 'Mooney Rivlin' 法の特殊なケースです。材料係数がひとつ少なくなります (C2 = 0)。デフォルトでは圧縮性のバージョンが使用されます。
- 'Ciarlet Geymonat': この法則には lambda、mu と gamma と呼ばれる 3 つのパラメータがあります。ここで gamma は]-lambda/2-mu, -mu[になるように選択されます。

材料法則のパラメータは、MeshFem *mf_d* で記述します。行列 *params* では、行はパラメータに対応し列は *nbdof(mf_d)* に対応します。

最後の引数は作成するもの、接線行列または RHS、を選択します。非圧縮性を考慮する場合、圧縮には MeshFem *mf_p* が必要です。

SpMat オブジェクト (接線行列)、vec オブジェクト (右辺)、SpMat オブジェクトのタプル (非圧縮接線行列)、または vec オブジェクトのタプル (非圧縮性右辺) を返します。

asm_helmholtz(*mim*, *mf_u*, *mf_d*, *k*, *region=None*)

Helmholtz 問題の行列の組み立て。

$\Delta u + k^2 u = 0$ 、ただし k は複素数スカラーです。

SpMat オブジェクトを返します。

asm_bilaplacian(*mim*, *mf_u*, *mf_d*, *a*, *region=None*)

Bilaplacian 問題のための行列の組み立てです。

$\Delta(a(x)\Delta u) = 0$ ただし *a* はスカラーです。

SpMat オブジェクトを返します。

asm_bilaplacian_KL(*mim*, *mf_u*, *mf_d*, *a*, *nu*, *region=None*)
Kirchhoff-Love 定式化を用いた Bilaplacian 問題に対する行列の組み立て。

$\Delta(a(x)\Delta u) = 0$ ただし *a* はスカラーです。

SpMat オブジェクトを返します。

asm_volumic_source(*mim*, *mf_u*, *mf_d*, *fd*, *region=None*)
体積ソース項のアセンブリ。

データ MeshFem *mf_d* で定義されたデータベクトル *fd* を使用して、MeshFem *mf_u* 上で組み立てられたベクトル *V* を出力します。 *fd* は実数値でも複素数値でもよいです。

vec オブジェクトを返します。

asm_boundary_source(*bnum*, *mim*, *mf_u*, *mf_d*, *G*)
境界ソース項の構築。

G は [Qdim x N] 行列でなければなりません。ここで N は *mf_d* の自由度数であり、(MeshFem の作成時に設定される)Qdim は未知の u の次元です。

vec オブジェクトを返します。

asm_dirichlet(*bnum*, *mim*, *mf_u*, *mf_d*, *H*, *R*, *threshold=None*)
h.u = r 型の Dirichlet 条件の集合です。

ハンドル *h.u = r* ここで h は (あらゆる rank の) 正方行列で、その大きさは未知の u の次元と等しいです。この行列は *H* 、 *mf_d* の自由度ごとに 1 列ずつ格納され、各列には行列 h の値が Fortran の次数で格納されます。

$$`H(:,j) = [h11(x_j)h21(x_j)h12(x_j)h22(x_j)]`$$

u が 2 次元ベクトルフィールドの場合。

もちろん、未知のスカラーフィールドである場合、単に *H = ones(1, N)* を設定する必要があります。ここで N は *mf_d* の自由度数です。

これは基本的に、 gf_asm('boundary qu term') を *H* に対して呼び出し、 gf_asm('neumann') を *R* に対して呼び出すのと同じですが、この関数は (可能な場合) ' より良い ' (より多くの対角線) 制約行列を生成しようとします。

Spmat.Dirichlet_nullspace() も参照してください。

asm_boundary_qu_term(*boundary_num*, *mim*, *mf_u*, *mf_d*, *q*)
境界 qu 項のアセンブリです。

q は [Qdim x Qdim x N] 配列でなければなりません。ここで N は *mf_d* の自由度数であり、Qdim は (MeshFem の作成時に設定される) 未知変数 u の次元です。

SpMat オブジェクトを返します。

asm_define_function(*name*, *nb_args*, *expression*, *expression_derivative_t=None*, **args*)

概要: asm_define_function(string name, int nb_args, string expression[, string expression_derivative_t[, string expression_derivative_u]])

高水準汎用アセンブリで使用できる新しい関数 *name* を定義します。関数には、1 つまたは 2 つのパラメータを指定できます。 *expression* では、汎用アセンブリの使用可能なすべての定義済み関数またはオペレーションを使用できます。ただし、一部の変数またはデータへの参照は指定できません。関数の引数は、1 つのパラメータ関数の場合は *t* 、2 つのパラメータ関数の場合は *t* と *u* です。例えば、 'sin(pi*t)+2*t*t' は 1 つのパラメータ関数のための有効な式であり、'sin(max(t,u)*pi)' は 2 つのパラメータ関数のための有効な式です。 *expression_derivative_t* と *expression_derivative_u* は *t* と *u* に関する導関数のオプション式です。指定されていない場合は、記号による微分が使用されます。

asm_undefine_function(*name*)

高水準の汎用アセンブリに対して以前に定義された関数 *name* の定義をキャンセルします。

asm_define_linear_hardening_function(*name*, *sigma_y0*, *H*, **args*)

概要: asm_define_linear_hardening_function(string name, scalar sigma_y0, scalar H, ... [string 'Frobenius'])

初期降伏応力が *sigma_y0* で硬化係数が H の新しい線形硬化関数を name という名前で定義します。値が 'Frobenius' の特別な文字列引数が指定された場合、硬化関数は Von-Mises 相当ではなく、入力ひずみと出力応力の Frobenius ノルムで表されます。

asm_define_Ramberg_Osgood_hardening_function(*name*, *sigma_ref*, **args*)

概要: asm_define_Ramberg_Osgood_hardening_function(string name, scalar sigma_ref, {scalar eps_ref | scalar E, scalar alpha}, scalar n[, string 'Frobenius'])

新しい Ramberg Osgood 硬化関数を、初期降伏応力を *sigma_y0* 、硬化係数を H として名前 *name* で定義します。値が 'Frobenius' の特別な文字列引数が指定されている場合、硬化関数は、Von-Mises 相当ではなく、入力ひずみと出力応力の Frobenius ノルムで表されます。

asm_expression_analysis(*expression*, **args*)

概要: asm_expression_analysis(string expression [, {Mesh mesh | MeshIm mim}] [, der_order] [, Model model] [, string varname, int is_variable[, {MeshFem mf | MeshImd mimd}], ...])

高水準汎用アセンブリ表現を解析し、指定された表現に関する情報を出力します。

asm_volumic(*CVLST=None*, **args*)

概要: (...) = asm_volumic(,CVLST], expr [, mesh_ims, mesh_fems, data...])

体積構築の低水準汎用アセンブリ 手順。

式 *expr* は引数 (オプションのデータ) にリストされた MeshFem に対して評価され、出力引数に割り当てられます。アセンブリ式の構文の詳細については、getfem ユーザマニュアル (か getfem++ のソースファイル getfem_assembling.h) を参照してください。

たとえば、フィールドの L2 ノルムは次のように計算できます。

```
gf_compute('L2 norm') or with the square root of:

gf_asm('volumic','u=data(#1); V()+=u(i).u(j).comp(Base(#1).Base(#1))(i,
↪j)',mim,mf,U)
```

Laplacian 剛性マトリックスは、次の式で評価できます。

```
gf_asm('laplacian',mim, mf, mf_data, A) or equivalently with:

gf_asm('volumic','a=data(#2);M(#1,#1)+=sym(comp(Grad(#1).Grad(#1).Base(
↪#2))(:,i,:,i,j).a(j))', mim,mf,mf_data,A);
```

asm_boundary (*bnum*, *expr*, *mim=None*, *mf=None*, *data=None*, **args*)

概要: (...) = asm_boundary(int bnum, string expr [, MeshIm mim, MeshFem mf, data...])

低水準の汎用境界のアセンブリです。

gf_asm('volumic') のヘルプを参照してください。

asm_interpolation_matrix (*mf*, **args*)

概要: Mi = asm_interpolation_matrix(MeshFem mf, {MeshFem mfi | vec pts})

MeshFem から別の MeshFem またはポイント集合の補間行列を構築します。

行列 *Mi* を返します。*V* = *Mi.U* は gf_compute('interpolate_on',mfi) と等しいです。反復補間に便利です。これは単なる補間であり、ここでは基本的な積分は行われず、 *mfi* は lagrangian でなければならないことに注意してください。より汎用的なケースでは、質量行列を介して L2 投影を行う必要があります。

Mi は SpMat オブジェクトです。

asm_extrapolation_matrix (*mf*, **args*)

概要: Me = asm_extrapolation_matrix(MeshFem mf, {MeshFem mfe | vec pts})

MeshFem から別の MeshFem またはポイントの集合に外挿行列を構築します。

行列 Me を返します。 *V* = *Me.U* は gf_compute('extrapolate_on',mfe) と等しいです。反復外挿に便利です。

Me は SpMat オブジェクトです。

asm_integral_contact_Uzawa_projection(*bnum*, *mim*, *mf_u*, *U*, *mf_lambda*, *vec_lambda*, *mf_obstacle*, *obstacle*, *r*, **args*)

概要: B = asm_integral_contact_Uzawa_projection(int bnum, MeshIm mim, MeshFem mf_u, vec U, MeshFem mf_lambda, vec vec_lambda, MeshFem mf_obstacle, vec obstacle, scalar r [, {scalar coeff | MeshFem mf_coeff, vec coeff} [, int option[, scalar alpha, vec W]]])

Uzawa アルゴリズムを用いて解くための具体的な構築手順です。 接触問題。 $-(lambda - r (u_N-g))_-$ の項を $lambda$ の有限要素空間に投影します。

vec オブジェクトを返します。

asm_level_set_normal_source_term(*bnum*, *mim*, *mf_u*, *mf_lambda*, *vec_lambda*, *mf_levelset*, *levelset*)

境界 *bnum* 上の *mf_u* 上で定義されたベクトルフィールドの levelset 関数 (levelset の法線) の勾配方向の成分と考えられる *mf_lambda* 上で *vec_lambda* で表されるソース項のアセンブリを実行します。

vec オブジェクトを返します。

asm_lsneuman_matrix(*mim*, *mf1*, *mf2*, *ls*, *region=None*)

levelset 行列のアセンブリ。

SpMat オブジェクトを返します。

asm_nlsgrad_matrix(*mim*, *mf1*, *mf2*, *ls*, *region=None*)

nlsgrad 行列のアセンブリ。

SpMat オブジェクトを返します。

asm_stabilization_patch_matrix(*mesh*, *mf*, *mim*, *ratio*, *h*)

安定化パッチ行列のアセンブリ。

SpMat オブジェクトを返します。

7.20 compute モジュール

compute_L2_norm(*MF*, *U*, *mim*, *CVids=None*)

(実数または複素数の) フィールド *U* の L2 ノルムを計算します。

CVids が指定されている場合、ノルムはリストされた要素でのみ計算されます。

compute_L2_dist(*MF*, *U*, *mim*, *mf2*, *U2*, *CVids=None*)

U と *U2* の間の L2 距離を計算します。

CVids が指定されている場合、ノルムはリストされた要素でのみ計算されます。

compute_H1_semi_norm(*MF*, *U*, *mim*, *CVids=None*)

grad(*U*) の L2 ノルムを計算します。

CVids が指定されている場合、ノルムはリストされた要素でのみ計算されます。

compute_H1_semi_dist(*MF*, *U*, *mim*, *mf2*, *U2*, *CVids=None*)

U と *U2* の間の準 H1 距離を計算します。

CVids が指定されている場合、ノルムはリストされた要素でのみ計算されます。

compute_H1_norm(*MF*, *U*, *mim*, *CVids=None*)

U の H1 ノルムを計算します。

CVids が指定されている場合、ノルムはリストされた要素でのみ計算されます。

compute_H2_semi_norm(*MF*, *U*, *mim*, *CVids=None*)

D^2(*U*) の L2 ノルムを計算します。

CVids が指定されている場合、ノルムはリストされた要素でのみ計算されます。

compute_H2_norm(*MF*, *U*, *mim*, *CVids=None*)

U の H2 ノルムを計算します。

CVids が指定されている場合、ノルムはリストされた要素でのみ計算されます。

compute_gradient(*MF*, *U*, *mf_du*)

MeshFem *mf_du*'上で定義されたフィールド 'U の勾配を計算します。

勾配は MeshFem の *mf_du* で補間され *DU* に返されます。例えば、 *U* が P2 MeshFem で定義されている場合、 *DU* は P1 不連続 MeshFem で評価されます。 *mf* と *mf_du* は同じメッシュを共有する必要があります。

U は任意の数の次元を持つことができます (すなわち、この関数はスカラー場の勾配に限定されず、テンソル場にも使用できます)。ただし、 *U* の最後の次元は *mf* の自由度の数に等しくなければなりません。たとえば、 *U* が [3x3xNmf] 配列 (ここで、Nmf は *mf* の自由度の数です) の場合、 *DU* は [Nx3x3[xQ]xNmf_du] 配列となり、 N はメッシュの次元、 Nmf_du は *mf_du* の自由度の数であり、 *Qdim_mf != Qdim_mf_du* の場合、オプションの Q 次が挿入されます、ここで、 Qdim_mf は *mf* の Qdim であり、Qdim_mf_du は *mf_du* の Qdim です。

compute_hessian(*MF*, *U*, *mf_h*)

MeshFem *mf_h* に定義されているフィールド *U* の Hessian を計算します。

gf_compute('gradient', MeshFem mf_du) も参照してください。

compute_eval_on_triangulated_surface(*MF*, *U*, *Nrefine*, *CVLIST=None*)

[廃止された機能! 将来のリリースで削除される予定です] 2 次元三角形メッシュ用に設計されたユーティリティ関数は補間された U 値を持つ三角形座標のリストを返します。これは、不連

続な高次要素上で定義されたデータを正確に視覚化するために使用できます。出力では、UP の 6 つの最初の行は三角形座標を含み、他の行は U(三角形の頂点ごとに 1 つ) の CVLIST の補間された値を含み、考慮されるべき凸包数のリストを示すことができ、もし使用されなければ、全てのメッシュ凸包が使用されます。U は行ベクトルでなければなりません。

compute_interpolate_on(*MF*, *U*, **args*)

概要: Ui = compute_interpolate_on(MeshFem MF, vec U, {MeshFem mfi | Slice sli | vec pts})

別の MeshFem、Slice、またはポイントのリストのフィールドを補間します。

- 別の MeshFem *mfi* 上の補間　*mfi* は Lagrangian でなければなりません。 *mf* と *mfi* が同じメッシュオブジェクトを共有する場合、補間は非常に高速になります。
- スライス *sli* の補間。　これは、洗練された P1-不連続メッシュの補間に似ていますが、はるかに高速です。また、 Slice('points') と組み合わせて使用し、特定のポイント集合におけるフィールド値を取得することもできます。
- 点 *pts* の集合の補間

gf_asm('interpolation matrix') も参照

compute_extrapolate_on(*MF*, *U*, *mfe*)

フィールドを別の MeshFem に外挿します。

mfe のメッシュが *mf* のメッシュに厳密に含まれている場合、この関数は gf_compute('interpolate_on') と厳密に同じ処理を行います。しかし、 *mfe* のメッシュが *mf* に正確に含まれていない場合 (曲線状のリファインされたメッシュと粗いメッシュの間の補間を想像してください)、 *mf* 外の値が外挿されます。

gf_asm('extrapolation matrix') も参照

compute_error_estimate(*MF*, *U*, *mim*)

事後誤差推定値を計算します。

現在、使用できるのは 1 つだけです。各凸包に対して、法線導関数のジャンプが面に対して積分されます。

compute_error_estimate_nitsche(*MF*, *U*, *mim*, *GAMMAC*, *GAMMAN*, *lambda_*, *mu_*, *gamma0*, *f_coeff*, *vertical_force*)

Nitsche 法の場合、事後誤差推定値を計算します。

現在、使用できるのは 1 つだけです。各凸包に対して、法線導関数のジャンプが面に対して積分されます。

compute_convect(*MF*, *U*, *mf_v*, *V*, *dt*, *nt*, *option=None*, **args*)

概要: compute_convect(MeshFem MF, vec U, MeshFem mf_v, vec V, scalar dt, int nt[, string option[, vec per_min, vec per_max]])

Characteristic-Galerkin 法を用いて定常状態の速度場 V に関する U の対流を計算します。結果は、そのまま *U* で返されます。この方法は、 U. *mf_v* が連続有限要素法を表現するための純粋な Lagrange 関数に限定されます。 *dt* は積分時間、 *nt* は特性上の積分ステップ数です。 *option* は、re-entrant 対流がある境界部分のオプションです。最も近い要素への外挿の場合は *option = 'extrapolation'* で、その境界上の定数値の場合は *option = 'unchanged'* です、周期境界の場合は *option = 'periodicity'* となります。後者のオプションでは、per_min、per_max の二つのベクトルが与えられ、周期領域 (per_max[k] < per_min[k] が設定されているコンポーネントに対しては、何も実行されません) の限界を表現しなければなりません。この方法はかなり散逸的ですが、安定しています。

7.21 delete モジュール

delete(*I*, *J=None*, *K=None*, **args*)

概要: delete(I[, J, K,...])

gf_mesh(), gf_mesh_im(), gf_slice() などで与えられる descriptor でなければなりません。

別のオブジェクトが I を使用している場合、両方のオブジェクトの削除が要求されたときにのみ、オブジェクト I が削除されます。

削除できるのは、 gf_workspace('stats') の出力にリストされているオブジェクトだけです (たとえば、gf_fem オブジェクトは破棄できません)。

gf_workspace('clear all') を使用して、すべてを一度に消去することもできます。

7.22 linsolve モジュール

linsolve_gmres(*M*, *b*, *restart=None*, **args*)

概要: X = linsolve_gmres(SpMat M, vec b[, int restart][, Mrecond P][,'noisy'][,'res', r][,'maxiter', n])

GMRES 法で *M.X = b* を解く。

任意で前処理として *P* を使用します。restart パラメーターのデフォルト値は 50 です。

linsolve_cg(*M*, *b*, *P=None*, **args*)

概要: X = linsolve_cg(SpMat M, vec b [, Mrecond P][,'noisy'][,'res', r][,'maxiter', n])

共役勾配法で *M.X = b* を解きます。

オプションで前処理 *P* を使用します。

linsolve_bicgstab(*M*, *b*, *P=None*, **args*)

概要: X = linsolve_bicgstab(SpMat M, vec b [, Mrecond P][,'noisy'][,'res', r][,'maxiter', n])

双共役勾配安定化法で *M.X = b* を解く。

オプションで前処理 *P* を使用します。

linsolve_lu(*M*, *b*)

gf_linsolve('superlu',...) のエイリアス

linsolve_superlu(*M*, *b*)

M.U = b を解くには、SuperLU ソルバ (疎 LU 分解) を適用します。

条件数推定値 *cond* は、解 *U* とともに返されます。

linsolve_mumps(*M*, *b*)

MUMPS ソルバーを使用して、 *M.U = b* を解きます。

7.23 poly モジュール

poly_print(*P*)

P の内容を表示します。

poly_product(*P*)

完成予定です...!

7.24 util モジュール

util_save_matrix(*FMT*, *FILENAME*, *A*)

Harwell-Boeing フォーマット (FMT='hb') または Matrix-Market フォーマット (FMT='mm') を使用して、疎行列を FILENAME という名前のファイルにエクスポートします。

util_load_matrix(*FMT*, *FILENAME*)

ファイルから疎行列をインポートします。

util_trace_level(*level=None*)

getfem++ ルーチンの冗長性を設定します。

通常、モデルブリックによって出力されるメッセージは、0 でトレースメッセージがないことを意味します (デフォルトは 3 です)。レベルが指定されていない場合は、現在のトレースレベルが返されます。

util_warning_level(*level*)

重要度の低い警告を getfem でフィルタリングします。

0 は警告なし、デフォルト のレベルは 3 です。レベルが指定されていない場合は、現在の警告レベルが返されます。

索引