CHomP 入門（発展編）

Paweł Pilarczyk

京都大学

（訳：平岡裕章，広島大学理学研究科）

平成 19 年 8 月 8 日

1　連続写像の誘導準同型写像

方体集合対 (X, A) と (Y, B) の間の連続写像 $f\colon (X, A) \to (Y, B)$ が与えられたとき，その誘導準同型写像は CHomP を用いて計算可能である．そこで使用されているアルゴリズムの詳細については [1] を参照されたい．このアルゴリズムは方体集合に付随する様々な特徴を用いており，単体複体間の連続写像に対する誘導準同型写像には適用できない．単体複体の誘導準同型写像に関しては効率的なアルゴリズムが現在のところ存在しておらず，このことは方体集合を基礎としたホモロジー群計算の有用性を意味している．

1.1　方体写像

方体集合 X, Y が与えられたとき連続写像 $f\colon X \to Y$ は多価写像 $\mathcal{F}\colon \mathcal{X} \multimap \mathcal{Y}$ を用いて表現可能である．ここで $\mathcal{X}, \mathcal{Y}$ はその実現が $X = |\mathcal{X}|, Y = |\mathcal{Y}|$ となる縮退していない基本方体（各辺の長さが全て 1）の集まりである．また $\mathcal{F}$ は $\mathcal{X}$ の各基本方体 Q に対して $\mathcal{Y}$ の基本方体の集まり $\mathcal{F}(Q) \subset \mathcal{Y}$ を対応させる多価写像である（$\multimap$ は多価写像を表す記号）．このような写像 $\mathcal{F}$ は組み合わせ的方体多価写像もしくは単に方体写像と呼ばれる．また全ての基本方体 $Q \in \mathcal{X}$ について $f(Q) \subset |\mathcal{F}(Q)|$ が成り立つとき $\mathcal{F}$ を f の表現と呼ぶ．

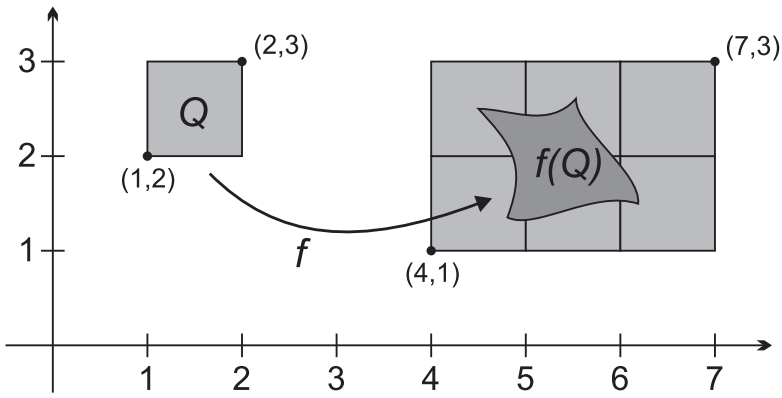

図 1: 基本方体 Q の f とその表現である方体写像 $\mathcal{F}$ による像

CHomP では方体写像を定めるテキストファイルに対して 2 通りの書式を用意している．1 つ目は異なる行ごとに対応 $\mathcal{X} \ni Q \mapsto \{Q_1, \ldots, Q_n\} \subset \mathcal{Y}$ を記入する方法である．ここで基本方体の集まりは空白で区切って指定される．例えば `(1,2) -> {(4,1) (4,2) (5,1) (5,2) (6,1) (6,2)}` は図 1 に示されている $\mathcal{F}\big([1,2] \times [2,3]\big) = \big\{[4,5] \times [1,2],\ [4,5] \times [2,3],\ [5,6] \times [1,2],\ [5,6] \times [2,3],\ [6,7] \times [1,2],\ [6,7] \times [2,3]\big\}$ を意味する．基本編でも説明しているが各基本方体は最も座標の値が小さい頂点で指定される．次のリストはこの書式に従った方体写像の例である．

例 **1** 書式 1

```
(0,0) -> {(0,0) (0,1)}
(0,1) -> {(0,1) (0,2) (0,3)}
(0,2) -> {(0,4) (1,4)}
(0,3) -> {(2,4) (3,4)}
(0,4) -> {(4,3) (4,4)}
(1,4) -> {(4,1) (4,2) (4,3)}
(2,4) -> {(3,0) (4,0)}
(3,4) -> {(1,0) (2,0)}
(4,4) -> {(0,0) (0,1)}
(4,3) -> {(0,1) (0,2) (0,3)}
(4,2) -> {(0,4) (1,4)}
(4,1) -> {(2,4) (3,4)}
(4,0) -> {(4,3) (4,4)}
(3,0) -> {(4,1) (4,2) (4,3)}
(2,0) -> {(3,0) (4,0)}
(1,0) -> {(1,0) (2,0)}
```

方体写像を定義する 2 つ目の書式は各基本方体 Q の像 $\mathcal{F}(Q)$ が凸集合になる場合に適用される．この場合最も座標が小さい頂点と大きい頂点を指定すれば凸集合が定まることに注意しよう．$\mathcal{X}$ の各基本方体も同様に表すと，この書式を用いた場合は図 1 の方体写像は `[(1,2)(2,3)] [(4,1)(7,3)]` で定められる．像が多くの基本方体からなる凸集合で与えられる場合には，その入力が書式 1 に比べて非常に簡単になる．ただし書式 2 の入力ファイルにはその先頭に付加的な情報（空間次元，$\mathcal{X}$ の構成要素数，写像の種類）を指定する必要がある．次の例 2 は書式 2 に従って記入された方体写像の例である．

例 **2** 書式 2

```
Space Dimension:  2
Number Of Primitive Arguments:  16
Map:  AlmostPerfect
Primitive Argument    Value
[(0,0) (1,1)]         [(0,0) (1,2)]
[(0,1) (1,2)]         [(0,1) (1,4)]
[(0,2) (1,3)]         [(0,4) (2,5)]
[(0,3) (1,4)]         [(2,4) (4,5)]
```

```
[(0,4) (1,5)]          [(4,3) (5,5)]
[(1,4) (2,5)]          [(4,1) (5,4)]
[(2,4) (3,5)]          [(3,0) (5,1)]
[(3,4) (4,5)]          [(1,0) (3,1)]
[(4,4) (5,5)]          [(0,0) (1,2)]
[(4,3) (5,4)]          [(0,1) (1,4)]
[(4,2) (5,3)]          [(0,4) (2,5)]
[(4,1) (5,2)]          [(2,4) (4,5)]
[(4,0) (5,1)]          [(4,3) (5,5)]
[(3,0) (4,1)]          [(4,1) (5,4)]
[(2,0) (3,1)]          [(3,0) (5,1)]
[(1,0) (2,1)]          [(1,0) (3,1)]
END
```

1.2 方体写像についての仮定

方体写像 $\mathcal{F}\colon (\mathcal{X},\mathcal{A}) \multimap (\mathcal{Y},\mathcal{B})$ は $\mathcal{F}(Q) \neq \emptyset$ や $\mathcal{F}(\mathcal{A}) \subset \mathcal{B}$ といった自明な仮定の他に，次で与えられる写像の非輪状性が満たされる必要がある．

定義 3 $\bigcap \mathcal{R} \neq \emptyset$ となる全ての基本方体の集まり $\mathcal{R} \subset \mathcal{X}$ に対して $|\mathcal{F}(\mathcal{R})|$ が非輪状となるとき，方体写像 $\mathcal{F}\colon \mathcal{X} \multimap \mathcal{Y}$ は非輪状であると呼ぶ．

この仮定は連続写像 f の表現 $\mathcal{F}$ を用いて誘導準同型写像 f_* を構成する為に必要となる [1]．

すぐわかるように対象としている方体写像がこの定義を満たしているかどうかを調べることは容易ではない．しかし書式 2 で扱ったように全ての基本方体の像 $\mathcal{F}(Q)$ が凸集合になっている場合には自動的にこの定義は満たされることが知られている．よって誘導準同型写像の計算にとっては各基本方体の像が凸集合となるような写像は扱いやすいことになる．

CHomP では与えられた方体写像が非輪状性を満たしているか確認するコマンド `chkmvmap` ("check multivalued map"の略) を提供している．このコマンドは次に紹介する `homcubes` と同様に用いられ，方体写像 `F.map`，方体集合 `X.cub`, `A.cub`, `Y.cub`, `B.cub` で与えられるテキストファイルに対して次のように用いられる．

```
chkmvmap F.map X.cub A.cub Y.cub B.cub
```

1.3 homcubes を用いた計算方法

基礎編にも出てきたコマンド `homcubes` を用いて非輪状性の仮定を満たす方体写像から誘導準同型写像を計算することができる．空間対の写像 $\mathcal{F}\colon (\mathcal{X},\mathcal{A}) \multimap (\mathcal{Y},\mathcal{B})$ を扱っている場合は $\mathcal{F}$, $\mathcal{X}$, $\mathcal{A}$, $\mathcal{Y}$, $\mathcal{B}$ を表す 5 つのテキストファイルが必要となる．$\mathcal{A} = \mathcal{B} = \emptyset$ の場合にはそれらを除いた 3 つのファイルが用いられる．例えば $H_*(X,A), H_*(Y,B)$ の生成元も保存する場合にはコマンド

```
homcubes F.map X.cub A.cub Y.cub B.cub -g XA.gen -g YB.gen
```

で誘導準同型写像が計算される．ここで XA.gen, YB.gen にはそれぞれのホモロジー群の生成元が保存されることになる．

1.4 Conley 指数への応用

連続写像 $f\colon (X,A) \to (Y,B)$（ここで $X \subset Y, A \subset B$）の誘導準同型写像を方体写像表現 $\mathcal{F}\colon (\mathcal{X},\mathcal{A}) \multimap (\mathcal{Y},\mathcal{B})$ を用いて計算する際，包含写像 $i\colon (X,A) \to (Y,B)$ がホモロジー群上の同型写像を誘導する場合がある．これは大雑把には (Y,B) は (X,A) を拡張したものと理解でき，離散力学系の Conley 指数の計算ではよく登場する．このような状況では $H_*(X,A)$ の生成元は $H_*(Y,B)$ の生成元としても扱われ，f と i^{-1} に対する誘導準同型写像の合成

$$(i_*)^{-1} \circ f_*\colon H_*(X,A) \to H_*(X,A)$$

が意味を持つことになる．

この合成写像は homcubes で-i オプションを付けることにより容易に計算される．実際には f_* と i_* の 2 つの誘導準同型写像を計算した後，i_* の逆写像を計算して 2 つを合成している．よって i_* が同型写像にならない場合には逆写像の計算時にエラーが出ることになる．

またこの場合，コマンド chkmvmap はオプション-i を付けることで写像 $f\colon (X,A) \to (Y,B)$ に対して $X \subset Y, A \subset B$ 等の非輪状性以外に必要になる仮定を検証してくれる．ただし i_* が同型写像であるかはこのオプションでは検証できず，homcubes を実行してみる必要がある．

2 C++インターフェース

CHomP ライブラリには C++言語で記述されたプログラム上でホモロジー計算関連の関数へアクセスを可能にするインターフェースが整備されている．より高度な数値計算の為にはこのようなインターフェースは必須であり，あらかじめデータを保存しその後 chomp や homcubes といったコマンドで計算させるといった手間が大幅に省ける．ここでは幾つかの基本的なインターフェースについて解説を与える．

2.1 基本インターフェース

基本インターフェースではコマンド chomp で用いられていたプログラムへの C++言語からのアクセスを可能にしている．ここで基本方体は各辺が 1 点に縮退していないものが扱われ，それらから構成される方体集合は bitmap 形式で用意する必要がある．ちなみに chomp でオプション指定をしていた engine はここでも選択可能になっている．

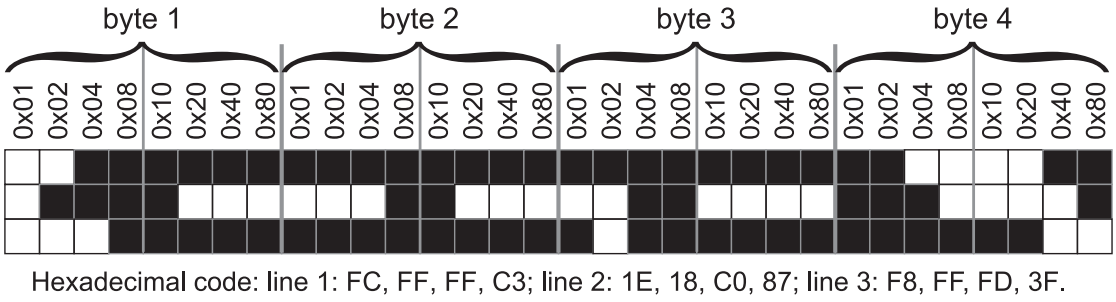

図 2: ビット列で表される bitmap 形式の例

2.1.1 Bitmap 形式

基本インターフェースでは方体集合を表すために bitmap 形式を採用しており，その n 次元 bitmap データは 1 次元配列として格納されることになる．ここでビット 1 は対応する場所に基本方体が存在しビット 0 は存在しないことを意味する．また取り扱う方体集合を内部に含む $\mathbf{R}^d$ 内の長方形領域 $[0 \times n_1] \times \cdots \times [0 \times n_d]$ をあらかじめ指定する必要があり，その為に整数 $n_1, \cdots, n_d$ を配列に格納することになる．その際技術的制約から n_1 の値は 32 ビット計算機では 32 の倍数，64 ビット計算機では 64 の倍数にする必要がある．各 1 バイトは 8 つの連続した基本方体に対応し，下位のビットほど座標の値が小さい基本方体を表す（図 2 を参照）．つまり bitmap データの最初のビットは基本方体 $(0, \cdots, 0)$ を表し，それに続く $n_1 - 1$ 個のビットが第 1 座標が 0 の基本方体を表すことになる．その後の n_1 個のビットは第 1 座標が 1 の基本方体に対応し，以下同様に各座標成分を大きくすることで d 次元方体集合が指定される．このように用意された方体集合に対してそのベッチ数を計算する関数は次で与えられる．

```
void ComputeBettiNumbers (const void *buffer, int *sizes, int dim,
int *result, const char *engine = 0, const int *wrapping = 0,
bool quiet = false);
```

次の C++プログラムでは `ComputeBettiNumbers` を使って図 2 で表された方体集合のベッチ数を計算させている．

プログラム 4 bitmap 形式で用意された方体集合のベッチ数計算

```
01 #include <iostream>
02 #include "capd/homengin/homology.h"
03
04 int main ()
05 {
06   const int dim = 2;
07   int sizes [] = {32, 3};
08   char buffer [] = {
09     '\xFC', '\xFF', '\xFF', '\xC3',
10     '\x1E', '\x18', '\xC0', '\x87',
11     '\xF8', '\xFF', '\xFD', '\x3F',
12   };
13
14   const char *engine = 0;
15   bool quiet = false;
16
```

```
17   int betti [dim + 1];
18   ComputeBettiNumbers (buffer, sizes, dim, betti, engine, quiet);
19   for (int i = 0; i <= dim; ++ i)
20     std::cout << (i ? " " : "") << betti [i];
21   std::cout << '\n';
22   return 0;
23 }
```

2.1.2 クラス CubicalSet

ComputeBettiNumbers を使うには bitmap 形式で方体集合を構成する必要があるが，その作成や操作を簡単に行う為に CHomP ではクラス CubicalSet が用意されている．CubicalSet のオブジェクトを作るには方体集合が定義される長方形領域

$$[n_1^-, n_1^+] \times \cdots \times [n_d^-, n_d^+]$$

を，d，$n_1^-, \ldots, n_d^-$，$n_1^+, \ldots, n_d^+$ で指定する必要がある．ここで用意された長方形領域に基本方体を加えるには関数 Add を，消去するには関数 Delete を用いればよい．これまでと同様に基本方体は座標の値が最小の頂点 $(k_1, \ldots, k_d)$ で表されるが，当然それらは不等式 $n_i^- \leq k_i < n_i^+$ を満たさなければならない．このようにして作られる方体集合のオブジェクトに対してそのベッチ数を計算させるには関数

```
void ComputeBettiNumbers (const CubicalSet &s,
int *result, const char *engine = 0, bool quiet = false);
```

を用いればよい．CubicalSet を用いたサンプルプログラムは以下で与えられる．

プログラム 5 クラス CubicalSet の使用例

```
01 #include <iostream>
02 #include "capd/homengin/cubiset.h"
03
04 int main ()
05 {
06   int left_coords [] = {-6, -5, 0};
07   int right_coords [] = {6, 1, 4};
08   CubicalSet Q (left_coords, right_coords, 3);
09
10   int cube1 [] = {1, -5, 0};
11   Q. Add (cube1);
12   int cube2 [] = {5, -2, 2};
13   Q. Add (cube2);
14
15   int betti [4];
16   ComputeBettiNumbers (Q, betti, "MM_CR", true);
17   for (int i = 0; i < 4; ++ i)
18     std::cout << (i ? " " : "") << betti [i];
19   std::cout << '\n';
20   return 0;
21 }
```

2.2 拡張インターフェース

拡張インターフェースでは一般の基本方体（辺が 1 点に縮退していてもよい），方体集合，方体複体，方体写像などに対応するクラスを用意している．またこれらのクラスに対

してホモロジー群を計算する関数も整っている．なおデータの入出力には通常の C++ と同様に << や >> を用いることができる．これらは名前空間 chomp::homology 内で使われることになる．

クラスのオブジェクトを指定するには頂点の座標を表す整数列を入力する必要があるが，標準では coordinate 型と呼ばれるものが用いられる（16 ビット整数 short int と同じ）．実用上は $[-32768, 32767]$ の整数で十分であると思われるが，それ以上の範囲使用する際はソースコードを参照されたい．

ホモロジー群 H_q は有限生成アーベル群であり

$$H_q \simeq \underbrace{\mathbf{Z} \oplus \cdots \oplus \mathbf{Z}}_{\beta_q} \oplus \mathbf{Z}_{p_1} \oplus \cdots \oplus \mathbf{Z}_{p_k}$$

の形で表現される．拡張インターフェース内では関数 BettiNumber や TorsionCoefficient を用いることでベッチ数 β_q やねじれ係数 $(p_1, \cdots, p_k)$ の情報が取り出せるようになっている．

2.2.1 クラス Cube, SetOfCubes

Cube は $\mathbf{R}^d$ 内で縮退していない基本方体を表す為に使われるクラスである．このクラスのコンストラクタは基本方体を指定する座標と次元の情報を必要とする．一方 Cube で指定されたオブジェクトから関数 coord によって座標の列，関数 dim で基本方体の次元が取り出せるようになっている．

このような基本方体で構成される方体集合を記述するクラスとして SetOfCubes がある．このクラスに基本方体を加えるには add 関数，消去するには remove 関数がそれぞれ用いられる．また与えられた基本方体が SetOfCubes のオブジェクトに既に含まれているかどうかを調べる関数 check も用意されている．

ここで紹介した SetOfCubes で表される方体集合 X や方体集合対 (X, A) のホモロジー群を計算する関数として Homology が用意されている．その使用例は Program 6 に示してある．(内容は Program 5 と同じ)．また Homology を用いた生成元や相対ホモロジー群の計算方法については Program 7，Program 8 を参照されたい．

プログラム 6 拡張インターフェースを用いたホモロジー群計算 (1)

```
01 #include <iostream>
02 #include "chomp/homology/homology.h"
03
04 using namespace chomp::homology;
05
06 int main ()
07 {
08   coordinate coords1 [] = {1, -5, 0};
09   Cube Q (coords1, 3);
10   SetOfCubes S;
11   S. add (Q);
12   coordinate coords2 [] = {5, -2, 2};
13   S. add (Cube (coords2, 3));
14
15   Chain *hom = 0;
16   int maxLevel = Homology (S, "S", hom);
```

```
17   for (int q = 0; q <= maxLevel; ++ q)
18     std::cout << (q ? " " : "") << BettiNumber (hom [q]);
19   std::cout << '\n';
20   delete [] hom;
21   return 0;
22 }
```

2.2.2 クラス CubicalCell, CubicalComplex

$\mathbf{R}^d$ 内の一般の基本方体を表すクラスとしては CubicalCell が用意されている．そのコンストラクタとしては縮退していない基本方体を作る為に Cube が必要とした入力や，縮退している場合に対応する最小と最大の頂点座標を表した整数列の対などが必要となる．

方体複体を表すクラスは CubicalComplex で与えられ，そのオブジェクトに基本方体を付け加えるには関数 add を用いる．与えられた方体複体 C の次元は関数 dim で参照でき，また指定された次元 k の基本方体の集まりを取り出すには [] を用いて C[k] とすればよい．ここで方体複体を定めるには全ての基本方体の面をあらかじめ CubicalComplex に入力しておく必要は無く，必要な面は計算の際に自動的に補間される．

次のサンプルプログラムは方体複体の構成からそのホモロジー群と生成元の計算までを示している．このプログラムではホモロジー群とその生成元を表示する際に関数 ShowHomology と ShowGenerators を用いている．

プログラム **7** 拡張インターフェースを用いたホモロジー群計算 (2)

```
01 #include <iostream>
02 #include "chomp/homology/homology.h"
03
04 using namespace chomp::homology;
05
06 int main ()
07 {
08   coordinate left [] = {1, 2, 0};
09   coordinate right [] = {2, 2, 1};
10   CubicalCell Q (left, right, 3);
11   CubicalComplex C;
12   C. add (Q);
13
14   Chain *hom = 0;
15   Chain **gen = 0;
16   int maxLevel = Homology (C, "C", hom, &gen);
17   ShowHomology (hom, maxLevel);
18   ShowGenerators (gen, hom, maxLevel);
19
20   delete [] hom;
21   for (int i = 0; i <= maxLevel; ++ i)
22     delete (gen [i]);
23   delete [] gen;
24   return 0;
25 }
```

2.2.3 クラス CubicalMap

最後に方体写像を表すクラス CubicalMap について説明しておく．これまでと同様に方体写像を扱う場合は全ての基本方体は縮退していないものとする．例えば Q を Cube のオブジェクト，F を CubicalMap のオブジェクトとすると，その像は SetOfCubes 型として F[Q] に格納される．像の指定にはこれまでにも出てきた基本方体を加える関数 add を用いるか，=を使って基本方体の集まりを直接与える方法がある．このようにして構成される CubicalMap の誘導準同型写像を計算するには関数 Homology を Program 8 のように用いればよい．

プログラム 8 拡張インターフェースを用いた誘導準同型写像の計算

```
01 #include <iostream>
02 #include "chomp/homology/homology.h"
03
04 using namespace chomp::homology;
05
06 int main ()
07 {
08   coordinate c1 [] = {1, 1, 0}, c2 [] = {1, 1, 1};
09   CubicalMap F;
10   SetOfCubes S;
11   S. add (Cube (c1, 3));
12   S. add (Cube (c2, 3));
13   Cube Q (c1, 3);
14   F [Q] = S;
15
16   SetOfCubes X = F. getdomain (), A, Y = S, B;
17   Chain *homX = 0, *homY = 0;
18   ChainMap *homF = 0;
19   int maxX = 0, maxY = 0;
20   Homology (F, X, A, Y, B, homX, maxX, homY, maxY, homF);
21
22   delete homF;
23   delete [] homX;
24   delete [] homY;
25   return 0;
26 }
```

参考文献

[1] K. Mischaikow, M. Mrozek, P. Pilarczyk, *Graph approach to the computation of the homology of continuous maps*, Foundations of Computational Mathematics 5 (2005) 199–229.