2 元分割表 HGM 関数

Risa/Asir 2 元分割表 HGM 関数説明書
1.0 版
2016 年 3 月 22 日

by Y.Goto, Y.Tachibana, N.Takayama

1 2 元分割表 HGM の関数説明書について

この説明書では HGM(holonomic gradient method) を用いた 2 元分割表の関数について説明する. ChangeLog の項目は www.openxm.org の cvsweb でソースコードを読む時の助けになる情報が書かれている.

本文中で引用している文献を列挙する.

- [GM2016] Y.Goto, K.Matsumoto, Pfaffian equations and contiguity relations of the hypergeometric function of type (k+1,k+n+2) and their applications, arxiv:1602.01637 (version 1)
- [T2016] Y.Tachibana, 差分ホロノミック勾配法のモジュラーメソッドによる計算の高速化, 2016, 神戸大学修士論文.
- [GTT2016] Y.Goto, Y.Tachibana, N.Takayama, 2 元分割表に対する差分ホロノミック勾配法の実装, 数理研講究録 (掲載予定).
- [TKT2015] N.Takayama, S.Kuriki, A.Takemura, A-hypergeometric distributions and Newton polytopes. arxiv:1510.02269

このマニュアルで説明する関数を用いたプログラム例は gtt_ekn/test-t1.rr など.

2 2 元分割表 HGM の関数

2.1 超幾何関数 E(k,n)

2.1.1 gtt_ekn.gmvector

`gtt_ekn.gmvector(`*beta*`,`*p*`)`
:: 周辺和 *beta*, セルの確率 p の二元分割表に付随する超幾何関数 E(k,n) の値およびその微分の値を戻す.

`gtt_ekn.ekn_cBasis_2(`*beta*`,`*p*`)`
の別名である.

return ベクトル, 超幾何関数の値とその微分. 詳しくは下記.

beta 行和, 列和のリスト. 成分はすべて正であること.

p 二元分割表のセルの確率のリスト

- gmvector は Gauss-Manin vector の略である [GM2016].
- gmvector の戻り値は [GM2016] の 6 章 p.23 のベクトル S である. これは [GM2016] の 4 章で定義されているベクトル F の定数倍であり, その定数は第一成分が [GM2016] の 6 章で定義されている級数 S の値と等しくなるように決められている.
- r1 x r2 分割表を考える. m+1=r1, n+1=r2 とおく. 正規化定数 Z は分割表 u を (m+1) × (n+1) 行列とするとき p^u/u! の和である. ここで和は行和列和が *beta* であるような u 全体でとる [TKT2015], [GM2016]. S はこの多項式 Z の p を

  ```
  [[1,y11,...,y1n],
   [1,y21,...,y2n],...,
   [1,ym1, ...,ymn],
   [1,1, ..., 1]]
  ```

  (1 が L 字型に並ぶ), と正規化した級数である.
- 2x(n+1) 分割表で, gmvector の戻り値を Lauricella F_D で書くことが以下のようにできる (b[2][1]-b[1][1] >= 0 の場合). ここで b[1][1], b[1][2] は, それぞれ 1 行目の行和, 2 行目の行和, b[2][i] は i 列目の列和である.

  ```
  S=F_D(-b[1,1], [-b[2,2],...,-b[2,n+1]], b[2,1]-b[1,1]+1 ; y)/C,
  ```

  C=b[1,1]! b[2,2]! ... b[2][n+1]! (b[2,1]-b[1,1])! とおく. 1/C は L 字型の分割表

  ```
  [[b[1,1],       0,       ..., 0       ],
   [b[2,1]-b[1,1],b[2,2], ..., b[2,n+1]]]
  ```

  に対応. gmvector は

  ```
  [S,(y11/a2) d_11 S,(y12/a3) d_12 S, ..., (y1n/a_(n+1)) d_1n S]
  ```

  である. ここで d_ij は yij についての微分,

  ```
    [a0,     a1, ...                          ,a_(n+2)]
  = [-b[1,2],-b[1,1],b[2,2], ..., b[2,n+1],b[2,1]]
  ```

  である.
- 周辺和 *beta* の時の正規化定数のセル確率 p に対する値は 多項式に退化した E(k,n) の値で表現できる. 文献 [TKT2015], [GM2016] 参照.

- option crt=1 (crt = Chinese remainder theorem) を与えると, 分散計算をおこなう [T2016]. 分散計算用の各種パラメータの設定は gtt_ekn.setup で行なう.

例: 次は 2 x 2 分割表で行和が [5,1], 列和が [3,3], 各セルの確率が [[1/2,1/3],[1/7,1/5]] の場合の gmvector の値である.

```
[3000] load("gtt_ekn.rr");
[3001] ekn_gtt.gmvector([[5,1],[3,3]],[[1/2,1/3],[1/7,1/5]])
[775/27783]
[200/9261]
```

参考: 2 x m 分割表 (Lauricella FD) についてはパッケージ tk_fd でも下記のように同等な計算ができる. 守備範囲の異なるプログラム同士の比較, debug 用参考.

```
[3080] import("tk_fd.rr");
[3081] A=tk_fd.marginal2abc([4,5],[2,4,3]);
[-4,[-4,-3],-1]  // 2 変数 FD のパラメータ. a,[b1,b2],c
[3082] tk_fd.fd_hessian2(A[0],A[1],A[2],[1/2,1/3]);
Computing Dmat(ca) for parameters B=[-4,-3],X=[ 1/2 1/3 ]
[4483/124416,[ 1961/15552 185/1728 ],
 [ 79/288 259/864 ]
 [ 259/864 47/288 ]]
// 戻値は [F=F_D, gradient(F), Hessian(F)]

// ekn_gt での例と同じパラメータ.
[3543] A=tk_fd.marginal2abc([5,1],[3,3]);
[-5,[-3],-1]
[3544] tk_fd.fd_hessian2(A[0],A[1],A[2],[(1/3)*(1/7)/((1/2)*(1/5))]);
Computing Dmat(ca) for parameters B=[-3],X=[ 10/21 ]
[775/27783,[ 20/147 ],[ 17/42 ]]
```

参考: 一般の A 分布の正規化定数についての Hessian の計算は実験的 package ot_hessian_ahg.rr で実装のテストがされている. (これはまだ未完成のテスト版なので出力形式等も将来的には変更される.)

```
import("ot_hgm_ahg.rr");
import("ot_hessian_ahg.rr");
def  htest4() {
  extern C11_A;
  extern C11_Beta;
  Hess=newmat(7,7);
  A =C11_A;
  Beta0= [b0,b1,b2,b3];
  BaseIdx=[4,5,6];
  X=[x0,x1,x2,x3,x4,x5,x6];
  for (I=0; I<7; I++) for (J=0; J<7; J++) {
    Idx = [I,J];
    H=hessian_simplify(A,Beta0,X,BaseIdx,Idx);
    Hess[I][J]=H;
    printf("[I,J]=%a, Hessian_ij=%a\n",Idx,H);
  }
```

```
  return(Hess);
}
[2917] C11_A;
[[0,0,0,1,1,1,1],[1,0,0,1,0,1,0],[0,1,1,0,1,0,1],[1,1,0,1,1,0,0]]
[2918] C11_Beta;
[166,36,290,214]
[2919] Ans=htest4$
[2920] Ans[0][0];
[[((b1-b0-1)*x4)/(x0^2),[4]],[((b1-b0-1)*x6)/(x0^2),[6]],
 [(b1^2+(-2*b0-1)*b1+b0^2+b0)/(x0^2),[]],[(x6)/(x0),[6,0]],[(x4)/(x0),[4,0]]]
```

参照 Section 2.1.5 [gtt_ekn.setup], page 8 ⟨undefined⟩ [gtt_ekn.pfaffian_basis], page ⟨undefined⟩

ChangeLog

- この関数は [GM2016] のアルゴリズムおよび [T2016] による modular method を用いた高速化を実装したものである.
- 変更を受けたファイルは OpenXM/src/asir-contrib/packages/src/gtt_ekn.rr 1.1, gtt_ekn/ekn_pfaffian_8.rr

2.1.2 gtt_ekn.nc

gtt_ekn.nc(*beta*,*p*)
:: 周辺和 *beta*, セルの確率 *p* の二元分割表の条件付き確率の正規化定数 Z およびその微分の値を戻す.

return ベクトル [Z,[[d_11 Z, d_12 Z, ...], ..., [d_m1 Z, d_m2 Z, ..., d_mn Z]]]

beta 行和, 列和のリスト. 成分はすべて正であること.

p 二元分割表のセルの確率のリスト

- r1 x r2 分割表を考える. m=r1, n=r2 とおく. 正規化定数 Z は分割表 u を m × n 行列とするとき p^u/u! の和である. ここで和は行和列和が *beta* であるような u 全体でとる [TKT2015], [GM2016]. p^u は p_ij^u_ij の積, u! は u_ij! の積である. d_ij Z で Z の変数 p_ij についての偏微分を表す.
- nc は gmvector の値を元に, [GM2016] の Prop 7.1 に基づいて Z の値を計算する.
- option crt=1 (crt = Chinese remainder theorem) を与えると, 分散計算をおこなう. 分散計算用の各種パラメータの設定は gtt_ekn.setup で行なう.

例: 2x3 分割表での Z とその微分の計算.

```
[2237] gtt_ekn.nc([[4,5],[2,4,3]],[[1,1/2,1/3],[1,1,1]]);
[4483/124416,[ 353/7776 1961/15552 185/1728 ]
[ 553/20736 1261/15552 1001/13824 ]]
```

参考: 2 x m 分割表 (Lauricella FD) についてはパッケージ tk_fd でも下記のように同等な計算ができる.

```
[3076] import("tk_fd.rr");
[3077] A=tk_fd.marginal2abc([4,5],[2,4,3]);
[-4,[-4,-3],-1]
[3078] tk_fd.ahmat_abc(A[0],A[1],A[2],[[1,1/2,1/3],[1,1,1]]);
```

```
RS=[ 4 5 ], CSnew=[ 2 4 3 ], Ynew=[ 1 1/2 1/3 ]
[ 1 1 1 ]
Computing Dmat(ca) for parameters B=[-4,-3],X=[ 1/2 1/3 ]
[4483/124416,[[353/7776,1961/15552,185/1728],
              [553/20736,1261/15552,1001/13824]]]
// 戻値は [Z, [[d_11 Z, d_12 Z, d_13 Z],
//             [d_21 Z, d_22 Z, d_23 Z]]] の値.
//           ここで d_ij は i,j 成分についての微分を表す.
```

参照 Section 2.1.5 [gtt_ekn.setup], page 8 Section 2.1.3 [gtt_ekn.lognc], page 5

ChangeLog

- 変更を受けたファイルは OpenXM/src/asir-contrib/packages/src/gtt_ekn.rr 1.1, gtt_ekn/ekn_eval.rr

2.1.3 gtt_ekn.lognc

`gtt_ekn.lognc(`*beta*`,`*p*`)`

:: 周辺和 *beta*, セルの確率 p の二元分割表の条件付き確率の正規化定数 Z の log の近似値およびその微分の近似値を戻す.

return ベクトル [log(Z), [[d_11 log(Z), d_12 log(Z), ...], [d_21 log(Z),...], ...]

beta 行和, 列和のリスト. 成分はすべて正であること.

p 二元分割表のセルの確率のリスト

- 条件付き最尤推定に利用する [TKT2015].
- option crt=1 (crt = Chinese remainder theorem) を与えると, 分散計算をおこなう. 分散計算用の各種パラメータの設定は gtt_ekn.setup で行なう.

例: 2 × 3 分割表での例. 第一成分のみ近似値.

```
[2238] gtt_ekn.lognc([[4,5],[2,4,3]],[[1,1/2,1/3],[1,1,1]]);
[-3.32333832422461674630,[ 5648/4483 15688/4483 13320/4483 ]
[ 3318/4483 10088/4483 9009/4483 ]]
```

参考: 2 x m 分割表 (Lauricella FD) についてはパッケージ tk_fd でも下記のように同等な計算ができる.

```
[3076] import("tk_fd.rr");
[3077] A=tk_fd.marginal2abc([4,5],[2,4,3]);
[-4,[-4,-3],-1]
[3078] tk_fd.log_ahmat_abc(A[0],A[1],A[2],[[1,1/2,1/3],[1,1,1]]);
RS=[ 4 5 ], CSnew=[ 2 4 3 ], Ynew=[ 1 1/2 1/3 ]
[ 1 1 1 ]
Computing Dmat(ca) for parameters B=[-4,-3],X=[ 1/2 1/3 ]
[-3.32333832422461674639485797719209322217260539267246045320,
 [[1.2598706, 3.499442, 2.971224],
  [0.7401293, 2.250278, 2.009591]]]
// 戻値は [log(Z),
//          [[d_11 log(Z), d_12 log(Z), d_13 log(Z)],
//           [d_21 log(Z), d_22 log(Z), d_23 log(Z)]]]
// の近似値.
```

参照 Section 2.1.5 [gtt_ekn.setup], page 8 Section 2.1.2 [gtt_ekn.nc], page 4

ChangeLog

- 変更を受けたファイルは OpenXM/src/asir-contrib/packages/src/gtt_ekn.rr 1.1.

2.1.4 gtt_ekn.expectation

`gtt_ekn.expectation(`*beta*`,`*p*`)`
:: 周辺和 *beta*, セルの確率 p の二元分割表の期待値を計算する.

return 二元分割表の各セルの期待値のリスト.

beta 行和, 列和のリスト. 成分はすべて正であること.

p 二元分割表のセルの確率のリスト

- [GM2016] の Algorithm 7.8 の実装.
- option crt=1 (crt = Chinese remainder theorem) を与えると, 分散計算をおこなう. 分散計算用の各種パラメータの設定は gtt_ekn.setup で行なう.
- option index を与えると, 指定された成分の期待値のみ計算する. たとえば 2 x 2 分割表で index=[[0,0],[1,1]] と指定すると, 1 のある成分の期待値のみ計算する.

2×2, 3×3 の分割表の期待値計算例.

```
[2235] gtt_ekn.expectation([[1,4],[2,3]],[[1,1/3],[1,1]]);
[ 2/3 1/3 ]
[ 4/3 8/3 ]
[2236] gtt_ekn.expectation([[4,5],[2,4,3]],[[1,1/2,1/3],[1,1,1]]);
[ 5648/4483 7844/4483 4440/4483 ]
[ 3318/4483 10088/4483 9009/4483 ]

[2442] gtt_ekn.expectation([[4,14,9],[11,6,10]],[[1,1/2,1/3],[1,1/5,1/7],[1,1,1]]);
[ 207017568232262040/147000422096729819 163140751505489940/147000422096729819
                                        217843368649167296/147000422096729819 ]
[ 1185482401011137878/147000422096729819 358095302885438604/147000422096729819
                                         514428205457640984/147000422096729819 ]
[ 224504673820628091/147000422096729819 360766478189450370/147000422096729819
                                        737732646860489910/147000422096729819 ]
```

参考: 2 x m 分割表 (Lauricella FD) についてはパッケージ tk_fd でも下記のように同等な計算ができる.

```
[3076] import("tk_fd.rr");
[3077] A=tk_fd.marginal2abc([4,5],[2,4,3]);
[-4,[-4,-3],-1]
[3078] tk_fd.expectation_abc(A[0],A[1],A[2],[[1,1/2,1/3],[1,1,1]]);
RS=[ 4 5 ], CSnew=[ 2 4 3 ], Ynew=[ 1 1/2 1/3 ]
[ 1 1 1 ]
Computing Dmat(ca) for parameters B=[-4,-3],X=[ 1/2 1/3 ]
[[5648/4483,7844/4483,4440/4483],
 [3318/4483,10088/4483,9009/4483]]
// 各セルの期待値.
```

参考: 一般の A 分布の計算は ot_hgm_ahg.rr. まだ実験的なため, module 化されていない. ot_hgm_ahg.rr についての参考文献: K.Ohara, N.Takayama, Pfaffian Systems of A-Hypergeometric Systems II — Holonomic Gradient Method, arxiv:1505.02947

```
[3237] import("ot_hgm_ahg.rr");
// 2 x 2 分割表.
[3238] hgm_ahg_expected_values_contiguity([[0,0,1,1],[1,0,1,0],[0,1,0,1]],
        [9,6,8],[1/2,1/3,1/5,1/7],[x1,x2,x3,x4]|geometric=1);
oohg_native=0, oohg_curl=1
[1376777025/625400597,1750225960/625400597,
 2375626557/625400597,3252978816/625400597]
// 2 x 2 分割表の期待値.

// 2 x 3 分割表.
[3238] hgm_ahg_expected_values_contiguity(
 [[0,0,0,1,1,1],[1,0,0,1,0,0],[0,1,0,0,1,0],[0,0,1,0,0,1]],
 [5,2,4,3],[1,1/2,1/3,1,1,1],[x1,x2,x3,x4,x5,x6]|geometric=1);
[5648/4483,7844/4483,4440/4483,3318/4483,10088/4483,9009/4483]
// 2 x 3 分割表の期待値. 上と同じ問題.
```

3 x 3 分割表. 構造的 0 が一つ.

```
/*
  dojo, p.221 のデータ. 成績 3 以下の生徒は集めてひとつに.
  2 1 1
  8 3 3
  0 2 6

  row sum: 4,14,8
  column sum: 10,6,10
  0 を一つ含むので, (3,6) 型の A から 7 列目を抜く.
*/

A=[[0,0,0,1,1,1, 0,0],
   [0,0,0,0,0,0, 1,1],
   [1,0,0,1,0,0, 0,0],
   [0,1,0,0,1,0, 1,0],
   [0,0,1,0,0,1, 0,1]];
B=[14,8,10,6,10];
hgm_ahg_expected_values_contiguity(A,B,[1,1/2,1/3,1,1/5,1/7,1,1],
                            [x1,x2,x3,x4,x5,x6,x7,x8]|geometric=1);

// 答.
[14449864949304/9556267369631,
 10262588586540/9556267369631, 13512615942680/9556267369631,
 81112808747006/9556267369631,
 21816297744346/9556267369631, 30858636683482/9556267369631,

 25258717886900/9556267369631,51191421070148/9556267369631]
```

3 x 3 分割表.

```
/*
 上のデータで 0 を 1 に変更.
  2 1 1
  8 3 3
  1 2 6

  row sum: 4,14,9
  column sum: 11,6,10
*/
A=[[0,0,0,1,1,1,0,0,0],
   [0,0,0,0,0,0,1,1,1],
   [1,0,0,1,0,0,1,0,0],
   [0,1,0,0,1,0,0,1,0],
   [0,0,1,0,0,1,0,0,1]];
B=[14,9,11,6,10];
hgm_ahg_expected_values_contiguity(A,B,[1,1/2,1/3,1,1/5,1/7,1,1,1],
                                   [x1,x2,x3,x4,x5,x6,x7,x8]|geometric=1);

// 期待値, 答.   x9 を指定していないので, 9 番目の期待値は出力してない.
[207017568232262040/147000422096729819,
 163140751505489940/147000422096729819,217843368649167296/147000422096729819,
 118548240101137878/147000422096729819,
 358095302885438604/147000422096729819,514428205457640984/147000422096729819,
 224504673820628091/147000422096729819,360766478189450370/147000422096729819]

// Z やその微分の計算は hgm_ahg_contiguity 関数がおこなうが, これの簡易インター
フェースは
// まだ書いてない.
```

参照 Section 2.1.5 [gtt_ekn.setup], page 8 Section 2.1.2 [gtt_ekn.nc], page 4

ChangeLog

- 変更を受けたファイルは OpenXM/src/asir-contrib/packages/src/gtt_ekn.rr 1.1.

2.1.5 gtt_ekn.setup

`gtt_ekn.setup()`
:: 分散計算用の環境設定をおこなう. 現在の環境を報告する.

return

- 使用するプロセスと素数の個数, 最小の素数を表示する. 準備されていない場合はその旨を表示.
- このパッケージでの分散計算は複数の cpu を搭載した計算機で実行されることを想定している.
- option nps (または number_of_processes) を与えると指定した数だけプロセスを用意する.
- option nprm (または number_of_primes) を与えると nprm が文字列の場合指定された素数リストのファイルを読み込む. nprm が自然数の場合さらに option minp (minp =MINimum Prime) を与えると minp より大きな素数を nprm 個生成する. その際 option fgp (または file_of_generated_primes) を与えると生成した素数リストをファイル名を fgp として保存する.

- 上記の option を指定しなかった場合次のデフォルト値が用いられる. nps=1. nprm=10. fgp=0.
- option report=1 を与えると現在の環境の報告のみを行う. setup(|report=1) の別名として report 関数を使用することもできる.

例: 素数のリストを生成してファイル p.txt へ書き出す.

```
gtt_ekn.setup(|nps=2,nprm=20,minp=10^10,fgp="p.txt")$
```

参照 Section 2.1.2 [gtt_ekn.nc], page 4 Section 2.1.1 [gtt_ekn.gmvector], page 2

ChangeLog

- 変更を受けたファイルは OpenXM/src/asir-contrib/packages/src/gtt_ekn.rr 1.1, gtt_ekn/g_mat_fac.rr

2.1.6 gtt_ekn.upAlpha

`gtt_ekn.upAlpha(`*i*`,`*k*`,`*n*`)`

::

i a_i を *a_i+1* と変化させる *contiguity relation.*
k E(k+1,n+k+2) 型の超幾何関数の *k.* 分割表では *(k+1)* × *(n+1).*
n E(k+1,n+k+2) 型の超幾何関数の *n.* 分割表では *(k+1)* × *(n+1).*
return contiguity relation の *pfaffian_basis* についての行列表現を戻す. *[GM2016]* の *Cor 6.3.*

- upAlpha は [GM2016] の Cor 6.3 の行列 U_i を戻す.
- 関連する各関数の簡潔な説明と例も加える.
- a_i を a_i-1 と変化させたい場合は関数 downAlpha を用いる.
- a_i と分割表の周辺和を見るには, 関数 marginaltoAlpha([行和, 列和]) を用いる.
- pfaffian_basis は [GM2016] の 4 章のベクトル F に対応する偏微分を戻す.

例: 以下の例は 2 × 2 分割表 (E(2,4)), 2 × 3 分割表 (E(2,5)) の場合である. [2225] までは出力を略している.

```
[2221] gtt_ekn.marginaltoAlpha([[1,4],[2,3]]);
[[a_0,-4],[a_1,-1],[a_2,3],[a_3,2]]
[2222] gtt_ekn.upAlpha(1,1,1);  // E(2,4) の a_1 方向の
                                //     contiguity を表現する行列
[2223] gtt_ekn.upAlpha(2,1,1);  // E(2,4) の a_2 方向
[2224] gtt_ekn.upAlpha(3,1,1);  // E(2,4) の a_3 方向
[2225] function f(x_1_1);
[2232] gtt_ekn.pfaffian_basis(f(x_1_1),1,1);
[ f(x_1_1) ]
[ (f1(x_1_1)*x_1_1)/(a_2) ]
[2233] function f(x_1_1,x_1_2);
f() redefined.
[2234] gtt_ekn.pfaffian_basis(f(x_1_1,x_1_2),1,2); // E(2,5), 2*3 分割表

[ f(x_1_1,x_1_2) ]
[ (f1,0(x_1_1,x_1_2)*x_1_1)/(a_2) ]
[ (f0,1(x_1_1,x_1_2)*x_1_2)/(a_3) ]
```

参照 Section 2.1.2 [gtt_ekn.nc], page 4 Section 2.1.1 [gtt_ekn.gmvector], page 2

ChangeLog

- この関数は [GM2016] で与えられたアルゴリズムに従い contiguity relation を導出する.
- 変更を受けたファイルはOpenXM/src/asir-contrib/packages/src/gtt_ekn/ekn_pfaffian_8.rr 1.1.

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

Table of Contents