

## 2 元分割表 HGM 関数

---

Risa/Asir 2 元分割表 HGM 関数説明書  
3.0 版  
2019 年 6 月 7 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](http://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. このパッケージは下記のようにロードする.

```
load("gtt_ekn3.rr");
```

gtt\_ekn3.rr は gtt\_ekn.rr を置き換える大きく改良されたパッケージである. 最新版の asir-contrib package を取得するには, 下記のように更新関数を呼び出す.

```
import("names.rr");
asir_contrib_update(|update=1);
```

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) (<http://arxiv.org/abs/1602.01637>)
- [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.
- [TGKT] Y.Tachibana, Y.Goto, T.Koyama, N.Takayama, Holonomic Gradient Method for Two Way Contingency Tables, arxiv:1803.04170 (the 2nd version) (<https://arxiv.org/abs/1803.04170>)
- [TKT2015] N.Takayama, S.Kuriki, A.Takemura,  $A$ -hypergeometric distributions and Newton polytopes. arxiv:1510.02269 (<http://arxiv.org/abs/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\_ekn3/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\_ekn3.gmvector

gtt\_ekn3.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返す.

gtt\_ekn3.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.
- $r1 \times r2$  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  $(m+1) \times (n+1)$  行列とすると  $p^u/u!$  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

$$\begin{aligned} & [[1, y_{11}, \dots, y_{1n}], \\ & [1, y_{21}, \dots, y_{2n}], \dots, \\ & [1, y_{m1}, \dots, y_{mn}], \\ & [1, 1, \dots, 1]] \end{aligned}$$

(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 ( $b[2][1]-b[1][1] \geq 0$  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.

$$\begin{aligned} S &= F_D(-b[1,1], [-b[2,2], \dots, -b[2,n+1]], b[2,1]-b[1,1]+1; y)/C, \\ C &= b[1,1]! b[2,2]! \dots b[2,n+1]! (b[2,1]-b[1,1])! \text{ とおく. } 1/C \text{ は L 字型の分割表} \\ & [[b[1,1], \quad 0, \quad \dots, 0], \\ & [b[2,1]-b[1,1], b[2,2], \dots, b[2,n+1]]] \end{aligned}$$

に対応. gmvector は

$$[S, (y_{11}/a_2) d_{11} S, (y_{12}/a_3) d_{12} S, \dots, (y_{1n}/a_{(n+1)}) d_{1n} S]$$

である. ここで  $d_{ij}$  は  $y_{ij}$  についての微分,

$$\begin{aligned} & [a_0, \quad a_1, \dots, \quad a_{(n+2)}] \\ & = [-b[1,2], -b[1,1], b[2,2], \dots, b[2,n+1], b[2,1]] \end{aligned}$$

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.

- 以下の option は expectation その他でも使える.
- 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\_ekn3.setup で行なう.
- option bs=1. binary splitting method で matrix factorial を計算. 一般に 3x3 では効果あり (assert2(15|bs=1)), 5x5 (test5x5(20|bs=1)) では遅くなる. デフォルトは bs=0.
- option path. contiguity を適用する path をきめるアルゴリズムを指定. path=2 (後藤, 松本の論文 [GM2016] の path). path=3 (論文 [TGKT] の path). デフォルトは path=3.
- option interval. 通常の matrix factorial の計算では, 分母と分子をそれぞれ整数計算で計算し最後に約分をする. しかしながら数の中間膨張が一般的に発生しその中間膨張を解消するため約分を一定間隔で行うと計算効率がよくなる. interval に整数値を設定することにより行列による一次変換を interval 回するたびに約分を行う. interval の最適値は問題毎に異なるためシステムがデフォルト値を設定することはない.
- option x=1. subprocess 毎に window を開く.

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_ekn3.rr");
[3001] gtt_ekn3.gmvector([[5,1],[3,3]], [[1/2,1/3],[1/7,1/5]])
[775/27783]
[200/9261]
```

例: N を 2 以上の自然数とする時, Gauss の超幾何関数(この場合は多項式となる)  $F(-36N, -11N, 2N, (1-1/N)/56)$  の値は T3 に代入される ( [TGKT] ).

```
N=2;
T2=gtt_ekn3.gmvector([[36*N,13*N-1],[38*N-1,11*N]], [[1,(1-1/N)/56],[1,1]])[0][0];
D=fac(36*N)*fac(11*N)*fac(2*N-1);
T3=T2*D;
```

ちなみに同じ値を Mathematica に計算させるには

```
n=2; Hypergeometric2F1[-36*n,-11*n,2*n,(1-1/n)/56]
```

例: interval option

```
[4009] P=gtt_ekn3.prob1(5,5,100);
[[[100,200,300,400,500],[100,200,300,400,500]], [[1,1/2,1/3,1/5,1/7],[1,1/11,1/13,1/17,1/19]]]

[4010] util_timing(quote(gtt_ekn3.gmvector([[100,200,300,400,500],[100,200,300,400,500]],
[cpu,72.852,gc,0,memory,4462742364,real,72.856]

[4011] util_timing(quote(gtt_ekn3.gmvector([[100,200,300,400,500],[100,200,300,400,500]],
[cpu,67.484,gc,0,memory,3535280544,real,67.4844]
```

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\_ekn3.setup], p. 9, `<undefined>` [gtt\_ekn3.pfaffian\_basis], p. `<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, [TGKT] の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
- interval option について変更を受けたファイルは OpenXM/src/asir-contrib/packages/src/gtt\_ekn3/ekn\_eval.1.6

### 2.1.2 gtt\_ekn3.nc

`gtt_ekn3.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返す.

*return*      ベクトル $[Z, [d_{11} Z, d_{12} Z, \dots], \dots, [d_{m1} Z, d_{m2} Z, \dots, d_{mn} Z]]$

*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

- $r_1 \times r_2$  分割表を考える.  $m=r_1, n=r_2$  とおく. 正規化定数  $Z$  は分割表  $u$  を  $m \times n$  行列とすると  $p^u/u!$  の和である. ここで和は行和列和が  $\beta$  であるような  $u$  全体とする [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_ekn3.setup` で行なう. その他の option は `gmvector` を参照.

例:  $2 \times 3$  分割表での  $Z$  とその微分の計算.

```
[2237] gtt_ekn3.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:  $2 \times m$  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\_ekn3.setup], p. 9, Section 2.1.3 [gtt\_ekn3.lognc], p. 6,

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\_ekn3.lognc

`gtt_ekn3.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返す.

*return*      ベクトル  $\log(Z)$ ,  $[[d_{11} \log(Z), d_{12} \log(Z), \dots], [d_{21} \log(Z), \dots], \dots]$

*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_ekn3.setup` で行なう.

例: 2 x 3 分割表での例. 第一成分のみ近似値.

```
[2238] gtt_ekn3.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_ekn3.setup`], p. 9, Section 2.1.2 [`gtt_ekn3.nc`], p. 5,

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\_ekn3.expectation

`gtt_ekn3.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. [TGKT] による高速化版(path=3) がデフォルト.
- 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\_ekn3.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.
- その他の option は gmvector を参照.

22, 33 の分割表の期待値計算例.

```
[2235] gtt_ekn3.expectation([[1,4],[2,3]],[[1,1/3],[1,1]]);
[ 2/3 1/3 ]
[ 4/3 8/3 ]
[2236] gtt_ekn3.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_ekn3.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,
 1185482401011137878/147000422096729819,
 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\_ekn3.setup], p. 9, Section 2.1.2 [gtt\_ekn3.nc], p. 5,

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\_ekn3.setup

gtt\_ekn3.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 fgpr (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 fgpr 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. fgpr=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.

- option subprogs=[file1,file2,...] により分散計算の子供プロセスにロードすべきファイル file1, file2, ... を指定する. default は subprogs=["gtt\_ekn3/childprocess.rr"] である.
- gtt\_ekn3.set\_debug\_level(Mode) で Ekn\_debug の値を設定する.

例: 素数のリストを生成してファイル p.txt へ書き出す.

```
gtt_ekn3.setup(|nps=2,nprm=20,minp=10^10,fgp="p.txt")$
```

例: chinese remainder theorem (crt) を使って gmvector を計算.

```
[2867] gtt_ekn3.setup(|nprm=20,minp=10^20);
[2868] N=2; T2=gtt_ekn3.gmvector([[36*N,13*N-1],[38*N-1,11*N]],
                                [[1,(1-1/N)/56],[1,1]] | crt=1)$
```

参照        Section 2.1.2 [gtt\_ekn3.nc], p. 5, Section 2.1.1 [gtt\_ekn3.gmvector], p. 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\_ekn3.upAlpha, gtt\_ekn3.downAlpha

```
gtt_ekn3.upAlpha(i,k,n)
gtt_ekn3.downAlpha(i,k,n)
::
```

$i$   $a_i$  を  $a_{i+1}$  ( $a_i$  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.
- optional 引数 arule, xrule で  $a_i$  や  $x_{i,j}$  を数にしたものをより効率的に求めることができる. 変化をうけるパラメータを数にしてしまっても特にエラー表示はしない.  $a_0$  で和の条件を調整しているので注意(ToDo, double check).

例: 以下の例は 22 分割表( $E(2,4)$ ), 23 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_ekn3.marginaltoAlpha([[1,4],[2,3]]);
      [[a_0,-4],[a_1,-1],[a_2,3],[a_3,2]]
[2222] gtt_ekn3.upAlpha(1,1,1); // E(2,4) の a_1 方向の
                                // contiguity を表現する行列
[2223] gtt_ekn3.upAlpha(2,1,1); // E(2,4) の a_2 方向
[2224] gtt_ekn3.upAlpha(3,1,1); // E(2,4) の a_3 方向
[2225] function f(x_1_1);
[2232] gtt_ekn3.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_ekn3.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) ]

[2235] RuleA=[[a_2,1/3],[a_3,1/2]]$ RuleX=[[x_1_1,1/5]]$
base_replace(gtt_ekn3.upAlpha(1,1,1),append(RuleA,RuleX))
-gtt_ekn3.upAlpha(1,1,1 | arule=RuleA, xrule=RuleX);

[ 0 0 ]
[ 0 0 ]

```

参照 Section 2.1.2 [gtt\_ekn3.nc], p. 5, Section 2.1.1 [gtt\_ekn3.gmvector], p. 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.

### 2.1.7 gtt\_ekn3.cmle

gtt\_ekn3.cmle(u) u を観測データとするととき,  $P(U=u \mid \text{row sum, column sum} = \text{these of } U)$  を最大化する, 各セルの確率の近似値を求める.

::

u 観測データ(分割表)

return セルの確率(分割表形式)

- u を観測データとするととき,  $P(U=u \mid \text{row sum, column sum} = \text{these of } U)$  を最大化する, 各セルの確率の近似値を求める.
- optional parameter で algorithm の振る舞い(たとえば有理数を近似して, 分母分子が小さい有理数にする, gradient descent の step 幅)を調整すべきだが, これは作業中. 2017.03.03

例: 2 x 4 分割表.

```

U=[[1,1,2,3],[1,3,1,1]];
gtt_ekn3.cmle(U);
[[ 1 1 2 3 ]
 [ 1 3 1 1 ],[[7,6],[2,4,3,4]], // Data, row sum, column sum
 [ 1 67147/183792 120403/64148 48801/17869 ] // probability obtained.
 [ 1 1 1 1 ]]
```

例: 上の例は次の関数に.

```
gtt_ekn3.cmle_test3();
```

参照 Section 2.1.4 [gtt\_ekn3.expectation], p. 6,

## ChangeLog

- gtt\_ekn3/mle.rr に本体がある.
- gtt\_ekn3.rr の cmle 関数は wrapper.

### 2.1.8 gtt\_ekn3.set\_debug\_level, gtt\_ekn3.show\_path, gtt\_ekn3.get\_svalue, gtt\_ekn3.assert1, gtt\_ekn3.assert2, gtt\_ekn3.assert3, gtt\_ekn3.probl

gtt\_ekn3.set\_debug\_level(*m*) debug メッセージのレベルを設定.  
 gtt\_ekn3.contiguity\_mat\_list\_2 使用する contiguity を構成.  
 gtt\_ekn3.show\_path() どのように contiguity を適用したかの情報.  
 gtt\_ekn3.get\_svalue() static 変数の値を得る.  
 gtt\_ekn3.assert1(*N*) 2x2 分割表で動作チェック.  
 gtt\_ekn3.assert2(*N*) 3x3 分割表で動作チェック.  
 gtt\_ekn3.assert3(*R1*, *R2*, *Size*) *R1* x *R2* 分割表で並列動作の動作チェック.  
 gtt\_ekn3.probl(*R1*,*R2*,*Size*) *R1* x *R2* 分割表用のテストデータを作る.  
 ::

*m* レベル.

- (*m* & 0x1) == 0x1 の時 g\_mat\_fac\_test\_plain と g\_mat\_fac\_itor の両方を呼び出し値を比較する(gtt\_ekn3.setup した状態で).
- (*m* & 0x2) == 0x2 の時 g\_mat\_fac\_test への引数を tmp-input-数.ab として保存.
- (*m* & 0x4) == 0x4 の時 matrix factorial の計算の呼び出し引数を表示.
- *N* は問題の周辺和のサイズ.
- get\_svalue の戻り値は[Ekn\_plist,Ekn\_IDL,Ekn\_debug,Ekn\_mesg,XRule,ARule,Verbose,Ekn\_Rq] の値.
- assert3 の options: x=1, subprocess の window を表示. nps=m, m 個のプロセスで contiguity を求める(contiguity\_mat\_list\_3). crt, interval などは gmvector などと共通の option. timing data を表示するには load("gtt\_ekn3/ekn-eval-timing.rr"); しておく.

例.

```
[2846] gtt_ekn3.set_debug_level(0x4);
[2847] N=2; T2=gtt_ekn3.gmvector([[36*N,13*N-1],[38*N-1,11*N]],
                                [[1,(1-1/N)/56],[1,1]])$
[2848] level&0x4: g_mat_fac_test([ 113/112 ]
[ 1/112 ],[ (t+225/112)/(t^2+4*t+4) (111/112*t+111/112)/(t^2+4*t+4) ]
[ (1/112)/(t^2+4*t+4) (111/112*t+111/112)/(t^2+4*t+4) ],0,20,1,t)
Note: we do not use g_mat_fac_itor. Call gtt_ekn3.setup(); to use the crt option.
level&0x4: g_mat_fac_test([ 67/62944040755546030080000 ]
[ 1/125888081511092060160000 ],[ (t+24)/(t^2+25*t+46) (2442)/(t^2+25*t+46) ]
[ (1)/(t^2+25*t+46) (-111*t-111)/(t^2+25*t+46) ],0,73,1,t)
level&0x4: g_mat_fac_test ----- snip
```

例.

```
[2659] gtt_ekn3.nc([[4,5,6],[2,4,9]],[[1,1/2,1/3],[1,1/5,1/7],[1,1,1]])$
[2660] L=matrix_transpose(gtt_ekn3.show_path())$
[2661] L[2];
```

```
----- snip
```

なお二番目の例の timing (total) [例では省略] は mod 計算を subprocess がやっている  
ので正しい値ではない. real time が計算時間の目安になる.

例.

3x5 分割表. 周辺和は 10 に比例する一定の数(factor option も関係. ソースを  
参照).

cell 確率は 1/素数で生成される.

```
[9054] L=gtt_ekn3.probl(3,5,10 | factor=1, factor_row=3);
[[[10,20,420],[30,60,90,120,150]], [[1,1/2,1/3,1/5,1/7],[1,1/11,1/13,1/17,1/19],[1,1,1,
[9055] number_eval(gtt_ekn3.expectation(L[0],L[1]));
[ 1.65224223218613 ... snip ]
```

例:

```
[5779] import("gtt_ekn3.rr"); load("gtt_ekn3/ekn_eval-timing.rr");
[5780] gtt_ekn3.assert3(5,5,100 | nps=32, interval=100);
-- snip
Parallel method: Number of process=32, File name tmp-gtt_ekn3/p300.txt is written.
Number of processes = 32.
-- snip
initialPoly of path=3: [ 2.184 0 124341044 2.1831 ] [CPU(s),0,*,real(s)]
contiguity_mat_list_3 of path=3: [ 0.04 0 630644 9.6774 ] [CPU(s),0,*,real(s)]
Note: interval option will lead faster evaluation. We do not use g_mat_fac_itor (crt).
g_mat_fac of path=3: [ 21.644 0 1863290168 21.6457 ] [CPU(s),0,*,real(s)]
Done. Saved in 2.ab
Diff (should be 0)=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,..., 0,0,0]
```

参照 Section 2.1.2 [gtt\_ekn3.nc], p. 5,

ChangeLog

- gtt\_ekn3/ekn\_eval.rr で matrix factorial の計算の呼び出し引数を表示する.
- grep 'iand(Ekn\_debug,0x1)' \*.rr でソースコードの該当の位置をさがす.



## 3 modular 計算

### 3.1 中国剰余定理と itor

#### 3.1.1 gtt\_ekn3.chinese\_itor

`gtt_ekn3.chinese_itor(data,idlist)`  
 :: mod p で計算した結果(ベクトル)から chinese remainder theorem, itor(integer to rational) で有理数ベクトルを得る.

*return* [val, n] ここで val は答え. また,  $n = n1 * n2 * \dots$

*data* [[val1,n1],[val2,n2], ...], ここで  $val \bmod n1 = val1$ ,  $val \bmod n2 = val2, \dots$

*idlist* chinese\_itor を実行するサーバ ID のリスト.

- 中国剰余定理を用いて  $val0 \bmod n1 = val1$ ,  $val0 \bmod n2 = val2, \dots$  となる  $val0$  を求める. val に algorithm itor を適用する.
- $\sqrt{n}$  より  $val0$  が大きい時は itor が適用されて  $val0$  が有理数  $val=a/b$  に変換される. つまり  $b*x=1 \bmod n$  となる逆数  $x$  を考えて,  $x*a \% n = val0$  となる数  $val$  を返す. 見つからないときは failure を返す.

例:  $[3!, 5^3 \cdot 3!] = [6, 750]$  が戻り値.  $6 \bmod 109 = 6$ ,  $750 \bmod 109 = 96$  が最初の引数の  $[[6,96],109]$ . 以下同様.

```
gtt_ekn3.setup(|nps=2,nprm=3,minp=101,fgp="p_small.txt");
SS=gtt_ekn3.get_svalue();
SS[0];
[103,107,109] // list of primes
SS[1];
[0,2] // list of server ID's
gtt_ekn3.chinese_itor([[[ 6,96 ],109],[[ 6,29 ],103],[[ 6,1 ],107]],SS[1]);
[[ 6 750 ],1201289]

// 引数はスカラーでもよい.
gtt_ekn3.chinese_itor([[96,109],[29,103]],SS[1]);
[[ 750 ],11227]
```

例: `gtt_ekn3/childprocess.rr` (server で実行される) の関数 `chinese` (chinese remainder theorem) と `euclid`.

```
load("gtt_ekn3/childprocess.rr");
chinese([newvect(2,[6,29]),103],[newvect(2,[6,750]),107*109]);
// mod 103 で [6,29], mod (107*109) で [6,750] となる数を mod 103*(107*109)
// で求めると,
[[ 6 750 ],1201289]
euclid(3,103); // mod 103 での 3 の逆数. つまり 1/3
-34
3*(-34) % 103; // 確かに逆数.
1
```

例: `gtt_ekn3/childprocess.rr` (server で実行される) の関数 `itor` (integer to rational) の例. `itor(Y,Q,Q2,Idx)` では  $Y < Q2$  なら  $Y$  がそのまま戻る. `Idx` は内部用の index で好きな数でよい. 戻り値の第 2 成分となる.

```
load("gtt_ekn3/childprocess.rr");
for (I=1;I<11; I++) print([I,itor(I,11,3,0)]);
[1,[1,0]]
[2,[2,0]]
[3,[-2/3,0]] //euclid(3,11); ->4, 4*(-2)%11 -> 3 なので確かに -2/3 は元
の数の候補
[4,[failure,0]]
[5,[-1/2,0]]
[6,[1/2,0]]
[7,[-1/3,0]]
[8,[failure,0]]
[9,[-2,0]]
[10,[-1,0]]
```

参照        Section 2.1.5 [gtt\_ekn3.setup], p. 9,

ChangeLog

- 関連ファイルは gtt\_ekn3/g-mat-fac.rr gtt\_ekn3/childprocess.rr

## 4 Binary splitting

## 4.1 matrix factorial

#### 4.1.1 gtt\_ekn3.init\_bsplrit, gtt\_ekn3.init\_dm\_bsplrit, gtt\_ekn3.setup\_dm\_bsplrit

```
gtt_ekn3.init_bsplrit(|minsize=16,levelmax=1);
:: binary split の実行のためのパラメータを設定する.
```

```
ggt_ekn3.init_dm_bsplrit(|bsplrit_x=0, bsplrit_reduce=0)
:: binary split の分散実行のためのパラメータを設定する.
```

```
gtt_ekn3.setup_dm_bsplrit(C)
:: binary split の分散実行のために C 個のプロセスを立ち上げる.
```

$C$  は  $levelmax-1$  に設定する. 特に  $levelmax=1$  のときは分散計算を行わない.

`bsplit_x=1` のとき, `debug` 用に各プロセスを `xterm` で表示.

- expectation などの関数に bs=1 オプションを与えると matrix factorial を binary splitting method で計算する.

例: bs=1 と無い場合の比較.

```
[4618] cputime(1)$
[4619] gtt_ekn3.expectation(Marginal=[[1950,2550,5295],[1350,1785,6660]],
                        P=[[17/100,1,10],[7/50,1,33/10],[1,1,1]]|bs=1)$
4.912sec(4.914sec)
[4621] V2=gtt_ekn3.expectation(Marginal=[[1950,2550,5295],[1350,1785,6660]],
                        P=[[17/100,1,10],[7/50,1,33/10],[1,1,1]])$
6.752sec(6.756sec)
```

例: 分散計算する場合. 分散計算はかえって遅くなる場合が多いので注意. 下記の例での `bsplit_x=1` option は debug windows を開くのでさらに遅くなる. `gtt_ekn3.test_bs_dist();` でもテストできる.

```
[3669] C=4$ gtt_ekn3.init_bsplrit(|minsize=16,levelmax=C+1)$ gtt_ekn3.init_dm_bsplrit(|b
[3670] [3671] [3672] gtt_ekn3.setup_dm_bsplrit(C);
[0,0]
[3673] gtt_ekn3.assert2(10|bs=1)$
```

参照 Section 2.1.1 [gtt\_ekn3.gmvector], p. 2, Section 2.1.4 [gtt\_ekn3.expectation], p. 6, `<undefined>` [gtt\_ekn3.assert1], p. `<undefined>`, `<undefined>` [gtt\_ekn3.assert2], p. `<undefined>`,

## ChangeLog

- 関連ファイルは `gtt_ekn3/mfac_bs.rr` `gtt_ekn3/dm_bs.rr`

## Index

(インデックスがありません)

(インデックスがありません)

## 簡単な目次

1	2 元分割表 HGM の関数説明書について .....	1
2	2 元分割表 HGM の関数 .....	2
3	modular 計算 .....	15
4	Binary splitting .....	17
	Index .....	18

# 目次

<b>1</b>	<b>2 元分割表 HGM の関数説明書について .....</b>	<b>1</b>
<b>2</b>	<b>2 元分割表 HGM の関数 .....</b>	<b>2</b>
2.1	超幾何関数 $E(k,n)$ .....	2
2.1.1	<code>gtt_ekn3.gmvector</code> .....	2
2.1.2	<code>gtt_ekn3.nc</code> .....	5
2.1.3	<code>gtt_ekn3.lognc</code> .....	6
2.1.4	<code>gtt_ekn3.expectation</code> .....	6
2.1.5	<code>gtt_ekn3.setup</code> .....	9
2.1.6	<code>gtt_ekn3.upAlpha</code> , <code>gtt_ekn3.downAlpha</code> .....	10
2.1.7	<code>gtt_ekn3.cmle</code> .....	11
2.1.8	<code>gtt_ekn3.set_debug_level</code> , <code>gtt_ekn3.show_path</code> , <code>gtt_ekn3.get_svalue</code> , <code>gtt_ekn3.assert1</code> , <code>gtt_ekn3.assert2</code> , <code>gtt_ekn3.assert3</code> , <code>gtt_ekn3.prob1</code> ..	12
<b>3</b>	<b>modular 計算 .....</b>	<b>15</b>
3.1	中国剰余定理と <code>itor</code> .....	15
3.1.1	<code>gtt_ekn3.chinese_itor</code> .....	15
<b>4</b>	<b>Binary splitting .....</b>	<b>17</b>
4.1	matrix factorial .....	17
4.1.1	<code>gtt_ekn3.init_bsplrit</code> , <code>gtt_ekn3.init_dm_bsplrit</code> , <code>gtt_ekn3.setup_dm_bsplrit</code> .....	17
	<b>Index .....</b>	<b>18</b>

