HGM functions for two way contingency tables.

HGM functions for two way contingency tables on Risa/Asir
Version 3.0
June 12, 2019

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

1 About this document

This document explains Risa/Asir functions for two way contingency tables by HGM(holonomic gradient method). Loading the package:

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

The package gtt_ekn3.rr is a major version up of gtt_ekn.rr. In order to download the latest asir-contrib package, please use the asir_contrib_update() as follows.

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

References cited in this document.

- [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, difference holonomic gradient method by the modular method. 2016, master thesis of Kobe University (in Japanese).
- [GTT2016] Y.Goto, Y.Tachibana, N.Takayama, implementation of difference holonomic gradient method for two way contingency tables. RIMS kokyuroku (in Japanese).
- [TGKT] Y.Tachibana, Y.Goto, T.Koyama, N.Takayama, Holonomic Gradient Method for Two Way Contingency Tables, arxiv:1803.04170 (the 3rd 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`)

The changelogs are described only in the Japanese version of this document.

2 Functions of HGM for two way contingency tables

2.1 Hypergeometric function E(k,n)

2.1.1 `gtt_ekn3.gmvector`

`gtt_ekn3.gmvector(`*beta*`,`*p*`)`

:: It returns the value of the hypergeometric function E(k,n) and its derivatives associated to the two way contingency table with the marginal sum *beta*, parameter *p* (cell probability).

It is an alias of gtt_ekn3.ekn_cBasis_2(*beta*,*p*)

return vector, see below.

beta a list of the row sum and the colum sum.

p the parameter.

- The name gmvector is an abreviation of the Gauss-Manin vector defined in [GM2016].
- The return value is the vector S in the page 23 (the section 6) of [GM2016]. This is a constant multiple of the vector F in the section 4 of [GM2016] and the constant is determined so that the first element of the vector is equal to the value of the series S in the section 6 of [GM2016].
- Consider an r1 x r2 contingency table. Put m+1=r1, n+1=r2. The normalizing constant Z is the sum of p^u/u! where u is an (m+1) x (n+1) matrix (contingency table) with non-negative integer entries. The sum is taken over u such that the row sum and the column sum of u are equal to *beta*, see [TKT2015], [GM2016], [TGKT]. The first element of S (polynomial in this case) is equal to this polynomial Z with a normalized p =

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

- The following options are also accepted by several functions, e.g., gmvector, expectation, nc.
- A distributed computation is turned on by the option crt=1 (crt = Chinese remainder theorem) [T2016]. The default is crt=0. Parameters for the distributed computation are set by gtt_ekn3.setup.
- Option bs=1. The matrix factorial, which is a product of contiguity relation matrices with different parameters, is evaluated by the binary splitting method. Examples: gtt_ekn3.assert2(15|bs=1)) (3x3 matrix), gtt_ekn3.test5x5(20|bs=1))(5x5 matrix). The default is bs=0.
- Option path. A choice of algorithms to apply contiguity relations. path=2 (the algorithm given in [GM2016]). path=3 (the algorithm given in [TGKT] (revised version)). The default is path=3.
- Option interval. The period of the intermediate reduction of numerators and denominators. A relevant value of ‘‘interval” will lead to an efficient evaluation, but no

optimal value of it is known. See [TGKT] as to details. The default is no intermediate reduction.

- Option x=1. It opens a window for each process.

Example: A 2 x 2 contingency table. The row sum is [5,1] and column sum is [3,3]. The parameter (cell probability) is [[1/2,1/3],[1/7,1/5]].

```
[3000] import("gtt_ekn3.rr");
[3001] gtt_ekn3.gmvector([[5,1],[3,3]],[[1/2,1/3],[1/7,1/5]])
[775/27783]
[200/9261]
```

Example: 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],
  [1,1/23,1/29,1/31,1/37],[1,1/41,1/43,1/47,1/53],[1,1,1,1,1]]]

[4010] util_timing(quote(gtt_ekn3.gmvector(P[0],P[1])[1];
[cpu,72.852,gc,0,memory,4462742364,real,72.856]

[4011] util_timing(quote(gtt_ekn3.gmvector(P[0],P[1]|interval=100)))[1];
[cpu,67.484,gc,0,memory,3535280544,real,67.4844]
```

`Refer to` Section 2.1.5 [gtt_ekn3.setup], page 5, ⟨undefined⟩ [gtt_ekn3.pfaffian_basis], page ⟨undefined⟩,

2.1.2 `gtt_ekn3.nc`

`gtt_ekn3.nc(`*beta*`,`*p*`)`
: :: It returns the normalizing constant Z and its derivatives for the two way contingency tables with the marginal sum *beta* and the parameter (cell probability) *p*. See, e.g., [TKT2015], [TGKT] as to the definition of Z.

return A list [Z,[[d_11 Z, d_12 Z, ...], ..., [d_m1 Z, d_m2 Z, ..., d_mn Z]]] where d_ij Z denotes the partial derivative of Z with respect to the parameter p_ij.

beta The row sum and the column sum.

p The parameter (cell probability).

- The function nc obtains Z from the value of gmvector by Prop 7.1 of [GM2016].
- See options for gmvector.

Example: A 2x3 contingency table.

```
[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 ]]
```

`Refer to` Section 2.1.5 [gtt_ekn3.setup], page 5, Section 2.1.3 [gtt_ekn3.lognc], page 4,

2.1.3 gtt_ekn3.lognc

`gtt_ekn3.lognc(`*beta*`,`*p*`)`
: :: It returns the logarithm of Z.

return A list [log(Z), [[d_11 log(Z), d_12 log(Z), ...], [d_21 log(Z),...], ...]

- This function is used to solve the conditional maximal likelihood estimation [TKT2015].
- See options of gmvector.

Example: A 2x3 contingency table. The first element is an approximate value of log(Z). The rests are exact values when the arguments of lognc are rational numbers.

```
[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 ]]
```

`Refer to` Section 2.1.5 [gtt_ekn3.setup], page 5, Section 2.1.2 [gtt_ekn3.nc], page 3,

2.1.4 gtt_ekn3.expectation

`gtt_ekn3.expectation(`*beta*`,`*p*`)`
: :: It returns the expectation of the hypergeometric distribution with the mariginal sum *beta* and the parameter *p*.

return The expectation of each cell.

- It is an implementation of Algorithm 7.8 of [GM2016]. A faster algorithm in [TGKT] is chosen with the default option path=3.
- By the option ‘‘index’’, it returns only the expections standing for the ‘‘index’’. For example, index=[[0,0],[1,1]] in the case of a 2 x 2 contingency table, it returns the expectations for the (2,1) and (2.2) elements (0 stands for no evaluation and 1 stands for doing the evaluation).
- See also options of gmvector.

Examples of the evaluation of expectations for 2 x 2 and 3 x 3 contingency tables.

```
[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 ]
```

`Refer to` Section 2.1.5 [gtt_ekn3.setup], page 5, Section 2.1.2 [gtt_ekn3.nc], page 3,

2.1.5 gtt_ekn3.setup

`gtt_ekn3.setup()`

:: It sets parameters for a distributed computation or report the current values of the parameters.

return 0

- It shows the number of processes, the number of primes, the minimal prime which is used.
- Option nps : the number of processes.
- Option nprm : the number of the primes used. When the argument of this option is a string, a list of primes are supposed to be given in the file by the name given by the string.
- Option minp : the minimal prime. It is used with the option nprm. It generates nprm primes more than or equal to minp. When the option fgp is given, the generated primes are stored in the file of the name fgp.
- The default values of nps, nprm, and fgp are nps=1. nprm=10. fgp=0 (no saving).
- The option report=1 shows the current parameters.
- Option subprogs=[file1,file2,...]. These files are loaded to the child processes. The default value is subprogs=["gtt_ekn3/childprocess.rr"].
- The function gtt_ekn3.set_debug_level(Mode) is used to set a debug message level (Ekn_debug)

Example: Generating a list of primes and outputing them to the file p.txt.

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

Example: Evaluating the gmvector by the Chinese remainder theorem (crt).

```
[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)$
```

`Refer to` Section 2.1.2 [gtt_ekn3.nc], page 3, Section 2.1.1 [gtt_ekn3.gmvector], page 2,

2.1.6 gtt_ekn3.upAlpha, gtt_ekn3.downAlpha

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

::

i It indicates the direction of the contiguity relation to get. In other words, the contiguity relation from a_i to a_i+1 (from a_i to a_i-1, the downAlpha case) is obtained.

k, n The contiguity relation for the hypergeometric function E(k+1,n+k+2) standing for the (k+1)(n+1) contingency table is obtained.

return The matrix representation of the contiguity relation with respect to the pfaffian_basis (see gtt_ekn3.pfaffian_basis). See also Cor 6.3 of [GM2016].

- The function upAlpha returns the matrix U_i of Cor 6.3 in [GM2016].

- The function downAlpha is for the contiguity relation from a_i to a_i-1 .
- The function marginaltoAlpha([row sum,column sum]) translates the marginal sum to values of a_i's.
- The function pfaffian_basis returns F in section 4 of [GM2016]. See the example below.
- The variables a_i and x_i_j can be specialized to numbers by the optional arguments arule and xrule. See the example below.

Example: 2x2 contingency table (E(2,4)), 2x3 contingency table (E(2,5)). Outputs of [2221] — [2225] are left out.

```
[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);  // contiguity relation of E(2,4)
                                //    for the a_1 direction
[2223] gtt_ekn3.upAlpha(2,1,1);  // E(2,4),  a_2 direction
[2224] gtt_ekn3.upAlpha(3,1,1);  // E(2,4),  a_3 direction
[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 contingency table
[ 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 ]
```

`Refer to` Section 2.1.2 [gtt_ekn3.nc], page 3, Section 2.1.1 [gtt_ekn3.gmvector], page 2,

2.1.7 `gtt_ekn3.cmle`

`gtt_ekn3.cmle(`*u*`)`
: :: It finds a parameter p (cell probability) which maximizes P(U=u | row sum, column sum = these of U) for given observed data u. The value of p is an approximate value.

u The observed data.

return An estimated parameter p

- Todo, optional parameter to set the step size of the gradient descent.

Example: 2x4 contingency table.

```
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 ]  // p obtained.
 [ 1 1 1 1 ]]
```

`Refer to` Section 2.1.4 [gtt_ekn3.expectation], page 4,

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.prob1`

`gtt_ekn3.set_debug_level(`*m*`)`
: :: It sets the level of debug messages.

`gtt_ekn3.contiguity_mat_list_2`
: :: It returns a list of contiguity directions to be used.

`gtt_ekn3.show_path()`
: :: It returns the path to apply contiguity relations. See [TGKT].

`gtt_ekn3.get_svalue()`
: :: It returns the values of the static variables.

`gtt_ekn3.assert1(`*N*`)`
: :: It checks the system by 2x2 contingency tables. *N* is proportional to the marginal sum.

`gtt_ekn3.assert2(`*N*`)`
: :: It checks the system by 3x3 contingency tables.

`gtt_ekn3.assert3(`*R1*`, `*R2*`, `*Size*`)`
: :: It checks the distributed computation system by R1 x R2 contingency tables.

`gtt_ekn3.prob1(`*R1*`,`*R2*`,`*Size*`)`
: :: It returns a test data for R1 x R2 contingency tables in the format [marginal sum, parameter p]. The marginal sum is proportional to *Size*. See benchmark tests in [TGKT].

- Let m be the debug level. When (m & 0x1) == 0x1, the values by g_mat_fac_test_plain and g_mat_fac_itor (distributed method is used) are compated. Note that gtt_ekn3.setup() is properly executed before doing these evaluations.
- When (m & 0x2) == 0x2, the arguments of g_mat_fac_test are stored in the file tmp-input-[number].ab.
- When (m & 0x4) == 0x4, the arguments for the matrix factorial computation are printed.
- The function `get_svalue` returns the list of the values of `[Ekn_plist,Ekn_IDL,Ekn_debug,Ekn_mesg,XRule,ARule,Verbose,Ekn_Rq]`.
- Options of assert3: “x=1” shows the window attached to every subprocess. With “nps=m”, m processes are used to obtain contiguity relations. The options crt, interval, ... of gmvector are also accepted. In order to display the timing data, do load("gtt_ekn3/ekn_eval-timing.rr"); before starting this function.

Example:

```
[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([ 67/6294404075546030080000 ]
[ 1/12588808151109206016000 ],[ (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
```

Example:

```
[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];
[2 1]
```

This means that the contiguity relations for the directions [2 1] (a_2, a_1) are used to evaluate the normalizing constant Z. L[0] is the contiguity matrix, L[1] is a list of the steps to apply for corresponding relations.

Example: Finding a path without evaluations of gmvectors.

```
A=gtt_ekn3.marginaltoAlpha_list([[400,410,1011],[910,411,500]])$
[2666] gtt_ekn3.contiguity_mat_list_2(A,2,2)$
[2667] L=matrix_transpose(gtt_ekn3.show_path())$
[2668] L[2];
[ 2 1 5 4 3 ]
[2669] gtt_ekn3.contiguity_mat_list_3(A,2,2)$ // new alg in [TGKT]
[2670] L=matrix_transpose(gtt_ekn3.show_path())$
[2671] L[2];
[2 1]  // shorter
```

Example: When assert2() returns 0 matrices, then the results of g_mat_fac_plain and g_mat_fac_int agree. In other words, the system is OK.

```
[8859] gtt_ekn3.assert2(1);
Marginal=[[130,170,353],[90,119,444]]
P=[[17/100,1,10],[7/50,1,33/10],[1,1,1]]
Try g_mat_fac_test_int: Note: we do not use g_mat_fac_itor. Call gtt_ekn3.setup(); to 
Timing (int) =0.413916 (CPU) + 0.590723 (GC) = 1.00464 (total), real time=0.990672

Try g_mat_fac_test_plain: Note: we do not use g_mat_fac_itor. Call gtt_ekn3.setup(); t
Timing (rational) =4.51349 (CPU) + 6.32174 (GC) = 10.8352 (total)
diff of both method =
[ 0 0 0 ]
[ 0 0 0 ]
[ 0 0 0 ]
```

```
[8860]

[8863] gtt_ekn3.setup(|nprm=100,minp=10^50);
Number of processes = 1.
Number of primes = 100.
Min of plist = 100000000000000000000000000000000000000000000000151.
0
[8864] gtt_ekn3.assert2(1 | crt=1);
Marginal=[[130,170,353],[90,119,444]]
P=[[17/100,1,10],[7/50,1,33/10],[1,1,1]]
Try [[crt,1]]
----  snip
```

Example: 3x5 contingency table. The parameter p (cell probability) is a list of 1/(prime number) 's.

```
[9054] L=gtt_ekn3.prob1(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 ]
```

Example:

```
[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,..., 0,0,0]
```

Refer to Section 2.1.2 [gtt_ekn3.nc], page 3,

3 Modular method

3.1 Chinese remainder theorem and itor

3.1.1 gtt_ekn3.chinese_itor

`gtt_ekn3.chinese_itor(`*data*`,`*idlist*`)`

:: It performs a rational reconstruction by the Chinese remainder theorem (itor = integer to rational).

return [val, n], the vector val is the value by the rational reconstruction. n = n1*n2*...

data [[val1,n1],[val2,n2], ...], val1, val2 are values evaluated in mod n1, mod n2, ... respectively. The relations val mod n1 = val1, val mod n2 = val2,.. are satisfied.

idlist The list of server id's for itor.

- When it cannot find val, it returns failure.

Example: [3!, 5^3*3!]=[6,750] is the return value. The relations 6 mod 109 =6, 750 mod 109=96 stand for [[6,96],109], ...

```
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]

// The argument may be a scalar.
gtt_ekn3.chinese_itor([[96,109],[29,103]],SS[1]);
  [[ 750 ],11227]
```

`Refer to` Section 2.1.5 [gtt_ekn3.setup], page 5,

4 Binary splitting

4.1 Matrix factorial

4.1.1 gtt_ekn3.init_bsplit, gtt_ekn3.init_dm_bsplit, gtt_ekn3.setup_dm_bsplit

`gtt_ekn3.init_bsplit(|minsize=16,levelmax=1);`
:: It sets parameters for the binary splitting to evaluate the matrix factorial M(1) M(2) ... M(n) where M(k) is a matrix with a parameter k.

`gtt_ekn3.init_dm_bsplit(|bsplit_x=0, bsplit_reduce=0)`
:: It sets parameters for the binary splitting by a distributed computation.

`gtt_ekn3.setup_dm_bsplit(C)`
:: It starts C processes for the binary splitting.

Option minsize.
When the size of the matrix factorial is less than the minsize, the binary splitting is not used and sequential multiplication is used instead.

Option levelmax.
The maximum of recursions of the recursive binary splitting in the distributed computation. See gtt_ekn3/dm_bsplit.rr C should be set to levelmax-1. When levalmax=1, the distributed computation is not performed.

Option bsplit_x.
When bsplit_x=1, a window attached to every process is opened.

Example: A comparison of bs=1 and no bs.

```
[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,P)$
6.752sec(6.756sec)
```

Example: Note that distributed computations are often slower than computations on a single process in our implementation of the binary splitting. The option bsplit_x=1 opens a debug windows, it makes things slower. The function gtt_ekn3.test_bs_dist() is a test function of the binary splitting by a distributed computation.

```
[3669] C=4$ gtt_ekn3.init_bsplit(|minsize=16,levelmax=C+1)$
gtt_ekn3.init_dm_bsplit(|bsplit_x=1)$
[3670] [3671] [3672] gtt_ekn3.setup_dm_bsplit(C);
[0,0]
[3673] gtt_ekn3.assert2(10|bs=1)$
```

`Refer to` Section 2.1.1 [gtt_ekn3.gmvector], page 2, Section 2.1.4 [gtt_ekn3.expectation], page 4, ⟨undefined⟩ [gtt_ekn3.assert1], page ⟨undefined⟩, ⟨undefined⟩ [gtt_ekn3.assert2], page ⟨undefined⟩,

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

Table of Contents