n_wishartd

n_wishartd User's Manual
Edition 1.0
Aug 2016

by Masayuki Noro

1 n_wishartd.rr

This manual explains about 'n_wishartd.rr', a package for computing a system of differential equations satisfied by the matrix 1F1 on a diagonal region. To use this package one has to load 'n_wishartd.rr'.

```
[...] load("n_wishartd.rr");
```

A prefix n_wishartd. is necessary to call the functions in this package.

1.1 Restriction of matrix 1F1 on diagonal regions

1.1.1 n_wishartd.diagpf

n_wishartd.diagpf(*m*,*blocks*)
: computes a system of PDEs satisfied by the *m* variate matrix 1F1 on a diagonal region specified by *blocks*.

return
: A list *[E1,E2,...]* where each *Ei* is a differential operator with partial fraction coefficients and it annihilates the restricted 1F1.

m
: A natural number

vars
: A list *[[s1,e1],[s2,e2],...]*.

options
: See below.

- This function computes a system of PDEs satisfied by the *m* variate matrix 1F1 on a diagonal region specified by *blocks*.
- Each component *[s,e]* in *blocks* denotes *ys=y(s+1)=...=ye*. The representative variable of this block is *ye*.
- One has to specify *blocks* so that all the variables appear in it. In particular a block which contains only one variable is specified by *[s,s]*.
- It has not yet been proven that this function always succeeds. At least it is known that this function succeeds if the size of each block <= 36.
- See Section 1.3 [Differential operators with partial fraction coefficients], page 4, for the format of the result.

```
[2649] Z=n_wishartd.diagpf(5,[[1,3],[4,5]]);
[
 [[[[-1,[]]],(1)*<<0,0,0,0,3,0>>],
  [[[-2,[[y1-y4,1]]],[-2,[[y4,1]]]],(1)*<<0,1,0,0,1,0>>],
  [[[9/2,[[y1-y4,1]]],[-3*c+11/2,[[y4,1]]],[3,[]]],(1)*<<0,0,0,0,2,0>>],
  ...
  [[[-6*a,[[y1-y4,1],[y4,1]]],[(4*c-10)*a,[[y4,2]]],[-4*a,[[y4,1]]]],
   (1)*<<0,0,0,0,0,0>>]],
 [[[[-1,[]]],(1)*<<0,4,0,0,0,0>>],

  [[[-6,[[y1-y4,1]]],[-6*c+10,[[y1,1]]],[6,[]]],(1)*<<0,3,0,0,0,0>>],
  [[[5,[[y1-y4,1]]],[-5,[[y1,1]]]],(1)*<<0,2,0,0,1,0>>],
  ...
```

```
      [[[21*a,[[y1-y4,2],[y1,1]]],[(36*c-87)*a,[[y1-y4,1],[y1,2]]],
       [-36*a,[[y1-y4,1],[y1,1]]],[(18*c^2-84*c+96)*a,[[y1,3]]],
       [-9*a^2+(-36*c+87)*a,[[y1,2]]],[18*a,[[y1,1]]]],(1)*<<0,0,0,0,0,0>>]]
    ]
```

1.1.2 n_wishartd.message

`n_wishartd.message(`*onoff*`)`
: starts/stops displaying messages during computation.

onoff : Start displaying messages if *onoff*=1. Stop displaying messages if *onoff*=0.

- This function starts/stops displaying messages during computation. The default value is set to 0.

1.2 Numerical comptation of restricted function

1.2.1 n_wishartd.prob_by_hgm

`n_wishartd.prob_by_hgm(`*m*`,`*n*`,`*[p1,p2,...]*`,`*[s1,s2,...]*`,`*t*`[|`*options*`])`
: computes the value of the distribution function of the largest eigenvalue of a Wishart matrix.

return

m : The number of variables.

n : The degrees of freedom.

[p1,p2,...,pk]
: A list of the multiplicities of repeated eigenvalues.

[s1,s2,...,sk]
: A list of repeated eigenvalues.

- Let *l1* be the largest eigenvalue of a Wishart matrix. Let *Pr[l1<t]* be the distribution function of *l1*. The function `n_wishartd.prob_by_hgm` computes the value of the distribution function by using HGM for a covariance matrix which has repeated eigenvalues *si* with multiplicity *pi* (*i=1,...,k*).
- This function repeats a Runge-Kutta method for the Pfaffian system by doubling the step size until the relative error between the current result and the previous result is less than *eps*, The default value of *eps* is *10^(-4)*.
- If an option *eq* is not set, a system of PDES satisfied by 1F1 on the diagonal region specified by *[p1,p2,...]* is computed. If an option *eq* is set, the list specified by *eq* is regarded as the correct system of PDEs.
- If an option *eps* is set, the value is used as *eps*.
- If an option *td* is set, the truncated power series solution for computing the initial vector is computed up to the total degree specified by *td*. The default value is 100.
- If an option *rk* is set, it is regarded as the order of a Runge-Kutta method. The default vaule is 5.

- It is recommended to use this function only when *k<=2* where *k* is the number of diagonal blocks because of the difficulty of the truncated power series solution and the difficulty of computation of the Pfaffian matrices.

```
[...] n_wishartd.message(1)$
[...] P=n_wishartd.prob_by_hgm(10,100,[9,1],[1/100,1],100|eps=10^(-6));
...
[x0=,8/25]
Step=10000
[0]
[8.23700622458446e-17,8.23700622459772e-17]
...
Step=1280000
[0][100000][200000][300000]...[900000][1000000][1100000][1200000]
[0.516246820120598,0.516246820227214]
[log ratio=,4.84611265040128]

Step=2560000
[0][100000][200000][300000]...[2200000][2300000][2400000][2500000]
[0.516246912003845,0.516246912217004]
[log ratio=,4.93705929488356]
[diag,18.6292,pfaffian,1.09207,ps,41.0026,rk,213.929]
0.516246912217004
266.4sec + gc : 8.277sec(276.8sec)
```

1.2.2 n_wishartd.prob_by_ps

`n_wishartd.prrob_by_ps(`*m*`,`*n*`,`*[p1,p2,...]*`,`*[s1,s2,...]*`,`*t*`[|`*options*`])`

computes the value of the distribution function of the largest eigenvalue of a Wishart matrix.

return

m The number of variables.

n The degrees of freedom.

[p1,p2,...,pk]
A list of the multiplicities of repeated eigenvalues.

[s1,s2,...,sk]
A list of repeated eigenvalues.

- This function compute a truncated power series solution up to a total degree where the relative error between the current value and the previous value at the desired point is less than *eps*. The default value of *eps* is *10^(-4)*. The value of the distribution function is computed by using this power series.
- If an option *eps* is set, the value is used as *eps*.
- If an option *eq* is not set, a system of PDES satisfied by 1F1 on the diagonal region specified by *[p1,p2,...]* is computed. If an option *eq* is set, the list specified by *eq* is regarded as the correct system of PDEs.

- It is recommened to use this function when t is small.

```
[...] Q=n_wishartd.prob_by_ps(10,100,[9,1],[1/100,1],1/2);
...
[I=,109,act,24.9016,actmul,0,gr,19.7852]
2.69026137621748e-165
61.69sec + gc : 2.06sec(64.23sec)
[...] R=n_wishartd.prob_by_hgm(10,100,[9,1],[1/100,1],1/2|td=50);
[diag,15.957,pfaffian,1.00006,ps,5.92437,rk,1.29208]
2.69026135182769e-165
23.07sec + gc : 1.136sec(24.25sec)
```

1.2.3 n_wishartd.ps

`n_wishartd.ps(`*z*`,`*v*`,`*td*`)`
: computes a truncated power series solution up to the total degree *td*.

return : A list of polynomial

z : A list of differential operators with partial fraction coefficients.

v : A list of variables.

- The result is a list *[p,pd]* where p is a truncated power series solution up to the total degree *td* and *pd* is the *td* homogeneous part of p.
- z cannot contain parameters other than the variables in v.

```
[...] Z=n_wishartd.diagpf(10,[[1,5],[6,10]])$
[...] Z0=subst(Z,a,(10+1)/2,c,(10+100+1)/2)$
[...] PS=n_wishartd.ps(Z0,[y1,y6],10)$
[...] PS[0];
19723078950274338395363 9/230438384724900975787223158176000*y1^10+
...
+(67388425421319768716722 33/10119537066347794279570342689043 20*y6^9
...+3932525/62890602*y6^2+1025/4181*y6+55/111)*y1
+19723078950274338395363 9/230438384724900975787223158176000*y6^10
+...+1395815/62890602*y6^3+3175/25086*y6^2+55/111*y6+1
```

1.3 Differential operators with partial fraction coefficients

1.3.1 Representation of partial fractions

The coefficients of the PDE satisfied by the matrix 1F1 are written as a sum of *1/yi* and *1/(yi-yj)* multiplied by constants. Furthermore the result of diagonalization by l'Hopital rule can also be written as a sum of partial fractions.

- A product *yi0^n0(yi1-yj1)^n1(yi2-yj2)^n2...(yik-yjk)^nk* in the denominator of a fraction is represented as a list *[[yi0,n0],[yi1-yj1,n1],...,[yik-yjk,nk]]*, Where each *yi-yj* satisfies *i>j* and the factors are sorted according to an ordering.
- Let f be a power sum as above and c a constant. Then a monomial *c/f* is represented by a list *[c,f]*. *f=[]* means that the denominator is 1.
- Finally *c1/f1+...+ck/fk* is represented as a list *[[c1,f1],...,[ck,fk]]*, where terms are sorted according to an ordering.

- We note that it is possible that a partial fraction is reduced to 0.

1.3.2 Representation of differential operators with partial fraction coefficients

By using partial fractions explained in the previous section, differential operators with partial fraction coefficients are represented. Let *f1,...,fk* be partial fractions and *d1,...,dk* distributed monomials such that *d1*>...>*dk*) with respected to the current monomial ordering. Then a differential operator *f1*d1*+...+fk*dk* is represented as a list *[f1,d1],...[fk,dk]]*.

1.3.3 Operations on differential operators with partial fraction coefficients

1.3.3.1 `n_wishartd.wsetup`

`n_wishartd.wsetup(m)`

m A natural number.

- This function sets a *m*-variate computational enviroment. The variables are *y0,y1,...,ym* and *dy0,...,dym*, where *y0, dy0* are dummy variables for intermediate computation.

1.3.3.2 `n_wishartd.addpf`

`n_wishartd.addpf(p1,p2)`

return A differential operator with partial fraction coefficients.

p1, p2 Differential operators with partial fraction coefficients.

- This function computes the sum of differential operators *p1* and *p2*.

1.3.3.3 `n_wishartd.mulcpf`

`n_wishartd.mulcpf(c,p)`

return A differential operator with partial fraction coefficients.

c A partial fraction.

p Differential operators with partial fraction coefficients.

- This function computes the product of a partial fraction *c* and a differential operator *p*.

1.3.3.4 `n_wishartd.mulpf`

`n_wishartd.mulpf(p1,p2)`

return A differential operator with partial fraction coefficients.

p1, p2 Differential operators with partial fraction coefficients.

- This function computes the product of differential operators *p1* and *p2*.

1.3.3.5 `n_wishartd.muldpf`

`n_wishartd.muldpf(y,p)`

return A differential operator with partial fraction coefficients.

y A variable.

p A differential operator with partial fraction coefficients.

- This function computes the product of the differential operator *dy* corresponding to a variable *y* and *p*.

```
[...] n_wishartd.wsetup(4)$
[...] P=n_wishartd.wishartpf(4,1);
[[[[1,[]]],(1)*<<0,2,0,0,0>>],[[[1/2,[[y1-y2,1]]],[1/2,[[y1-y3,1]]],
...,[[[-a,[[y1,1]]]],(1)*<<0,0,0,0,0>>]]
[...] Q=n_wishartd.muldpf(y1,P);
[[[[1,[]]],(1)*<<0,3,0,0,0>>],[[[1/2,[[y1-y2,1]]],[1/2,[[y1-y3,1]]],
...,[[[a,[[y1,2]]]],(1)*<<0,0,0,0,0>>]]
```

1.4 Experimental implementation of Runge-Kutta methods

In the function `n_wishartd.ps_by_hgm`, after computing the Pfaffian matrices for the sytem of PDEs on a diagonal region, it executes a built-in function `rk_ratmat` which computes an approximate solution of the Pfaffian system by Runge-Kutta method for a spcified step size. This function is repeated until the result gets stabilized, by doubling the step size. `rk_ratmat` can be used as a general-purpose Runge-Kutta driver and we explain how to use it.

1.4.1 rk_ratmat

`rk_ratmat(`*rk45*`,`*num*`,`*den*`,`*x0*`,`*x1*`,`*s*`,`*f0*`)`
solves a system of linear ODEs with rational function coefficients.

return A list of real numbers.

rk45 4 or 5.

num An array of constant matrices.

den A polynomial.

x0, x1 Real numbers.

s A natural number.

f0 A real vector.

- Let *k* be the size of an array *num*. The function `rk_ratmat` solves an initial value problem *dF/dx = P(x)F, F(x0)=f0* for *P(x)=1/den(num[0]+num[1]x+...+num[k-1]x^(k-1))* by a Runge-Kutta method.
- *rk45* specifies the order of a Runge-Kutta method. Adaptive methods are not implemented.
- The step size is specified by *s*. The step width is *(x1-x0)/s*.
- If the size of *f0* is *n*, each component of *num* is a square matrix of size *n*.
- The result is a list of real numbers *[r1,...,rs]* of length *s*. *ri* is the 0-th component of the solution vector after the step *i*. Before going to the next step the solution vector is divided by *ri*. Therefore the 0-th component of the final solution vector [varF(x1) is equal to *rs*r(s-1)*...*r1*.

- Since the ODE is linear, each step of Runge-Kutta method is also linear. This enables us to apply a normalization such that the 0-th component of each intermediate solution vector is set to 1. By applying this normalization we expect that all the components of intermediate solution vectors can be represented by the format of double precision floating point number. If there exist some components which cannot be represented by the format of double precision floating point number in the initial vector *f0*, we apply this normalization to *f0*. After applying `rk_ratmat` we multiply the result for the normalized *f0* and the 0-th component of the original *f0* to get the desired result.

```
[...] F=ltov([sin(1/x),cos(1/x),sin(1/x^2),cos(1/x^2)]);
[ sin((1)/(x)) cos((1)/(x)) sin((1)/(x^2)) cos((1)/(x^2)) ]
[...] F0=map(eval,map(subst,F,x,1/10));
[ -0.54402111088937 -0.839071529076452 -0.506365641109759 0.862318872287684 ]
[...] N0=matrix(4,4,[[0,0,0,0],[0,0,0,0],[0,0,0,-2],[0,0,2,0]])$
[...] N1=matrix(4,4,[[0,-1,0,0],[1,0,0,0],[0,0,0,0],[0,0,0,0]])$
[...] N=ltov([N0,N1])$
[...] D=x^3$
[...] R=rk_ratmat(5,N,D,1/10,10,10^4,F0)$
[...] for(T=R,A=1;T!=[];T=cdr(T))A *=car(T)[1];
[...] A;
0.0998334
[...] F1=map(eval,map(subst,F,x,10));
[ 0.0998334166468282 0.995004165278026 0.00999983333416666 0.999950000416665 ]
```

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

Table of Contents