CMAT© Newsletter: December 2011

Wolfgang M. Hartmann

December 2011

Contents

1 General Remarks

There were quite a number of problems with the `mrand()` and `kstest()` functions. I hope most of it was fixed and is now easier to understand. Maybe the related illustration at the end of this newsletter is even more helpful. For some reason I had never documented the `adtest()` and `adprob()` functions. Reviewing the paper by C. C. Wang forced me to test and think over those developments which were added stepwise in the past and mostly under time constraints.
Reviewing a paper by J. D. Richardson for JSS drew my attention to some newer uniform random generators:

- Mersenne-Twister (Matsumoto & Nishimura, 1998)
- Advanced Encryption Standard (AES) (Hellakalek & Wegenkittel, 2003)
- GFSR4 RNG (Ziff, 1998)
- RANLUX RNG (Lüscher, 1994)

Even though that this code was available on the internet it took quite some time to rewrite them in such a way that the methods could be used for CMAT. Testing my implementation of random generators I found that specifying `seed` values in hexadecimal form would be helpful, and I added that form of input to CMAT.
To cover more functionality of the SAS Enterprise Miner software I added the functions `assoc()`, `seq()`, and `rules()`. Another illustration at the end of this newsletter now shows that CMAT now covers a large part of that expensive software product and also shows what is still missing (some regression tree and neural nets methods).
More and more that LAPACK runtime option was used for a second choice of algorithm which was not even part of LAPACK and the name of the option was misleading. Now, the LAPACK option was changed to the SECOND option and the NOLAPACK option was changed to the PRIME option.
For the end of this year I am now preparing a newly tested release of CMAT. Just running the benchmarks shows that the number of bugs has increased and a major testing and bug fixing is necessary. I may even have a running Linux version then.

1.1 New Functions

assoc compute all associations tupels of items (for data mining: purchases of customers)

chngtxt change pattern in string data

convhull compute convex hull in d dimensions (also computes half space analysis) (Barber, Dobkin, & Hubdanpaa, 1996)

delaunay compute Delaunay triangulation in d dimensions (Barber, Dobkin, & Hubdanpaa, 1996)

cperm permute the columns of a matrix w.r.t. a given or random permutation

hankel create Hankel matrix

mvntest implements various tests for multivariate normality, like:

- Mardia's tests for multivariate skewness
- Mardia's tests for multivariate kurtosis
- Mardia & Foster (1983) omnibus test for skewness and kurtosis
- Royston (1983) W test (extension of Shapiro-Wilks test)
- $W_{min}(5)$ test by Wang and Hwang (2011)
- Henze-Zirkler (1990) test
- Small's $Q1$, $Q2$, $Q3$
- Doornik-Hansen Omnibus test
- Szekely-Rizzo test

rperm permute the rows of a matrix w.r.t. a given or random permutation

rules generate rules for shopping items (for data mining: create rules from results of `assoc()`)

sequ compute sequence tupels of items (for data mining: purchases of customers with a specified time component)

voronoi compute Voronoi diagrams in d dimensions (Barber, Dobkin, & Hubdanpaa, 1996)

1.2 Fixed Bugs

A number of bugs were fixed. There were many problems with the functions testing univariate normality which had to do with the fact that those functions were stepwise enhanced over the time and the changes never really very well tested.
Also quite a number of functions ignored the runtime option which is sets the uniform random generator. Until the end of the year some kind of well tested release of CMAT will be available.

2 Modifications of Features

None until now

3 Extensions to the Language

3.1 Use Hexadecimal Constants

The language (Yacc and Lex script) was extended for the input of hexadecimal numbers. They are converted into long integers and therefore must not be too big. Hexadecimal constants must begin with `0x` and may contain the digits $0, \ldots, 9, a, b, c, d, e, f$.

```
h1 = 0xabc1234;
print "Hexadecimal(0xabc1234) =",h1;
h2 = 0x2fddddd3;
print "Hexadecimal(0x2fddddd3)=",h2;
```

```
Hexadecimal(0xabc1234) = 180097588
Hexadecimal(0x2fddddd3)= 803069395
```

We found hexadecimal constants especially important for setting some specific seeds for uniform random generators:

1. Single source base 256 integer generator with single 256 digit deck (Richardson, 2011)

```
print "ISS1: Seed default: 0xabc1234: hexadecimal";
srand(0xabc1234,"iss1");
mu3 = rand(nr,1,'g',"iss1");
print "ISS1=", mu3;
```

```
ISS1: Seed default: 0xabc1234: hexadecimal

ISS1=
   |             1
------------------
 1 |     982600291
 2 |     735314717
 3 |    4087362757
 4 |    1802693449
 5 |    3799537722
 6 |     135030984
```

```
 7 |    3729099871
 8 |    2704712113
 9 |     349760511
10 |    2529659426
```

2. Twin source base 256 integer generator with four 65536 digit decks (Richardson, 2011)

```
print "ITS4: Seed default: 0x2fddddd3: hexadecimal";
srand(0x2fddddd3,"its4");
mu4 = rand(nr,1,'g',"its4");
print "ITS4=", mu4;
```

```
ITS4: Seed default: 0x2fddddd3: hexadecimal

ITS4=
   |             1
------------------
 1 |    3647235203
 2 |    1095945229
 3 |    3446603839
 4 |    1324252004
 5 |    3128063493
 6 |     638329403
 7 |    1752226044
 8 |    2687002194
 9 |     895754293
10 |    3862544476
```

4 Extensions to Various Functions

4.1 Extension of the `replace()` Function

The current form of the `replace()` function

b = replace(a,old,new<,rel>)

where the input arguments `old` and `new` are scalars:

a The first argument is a scalar, vector, or matrix object for which some of its entries should be changed.

old The second argument is the scalar value with which the entries of the first argument `a` are compared for a possible replacement.

new The third argument is the scalar value which replaces specific entries of the input object.

rel The (optional) fourth argument specifies one of the comparison relationships $=$, $\leq$, $\geq$, $<$, or $>$.

is extended to the following three situations:

old	new	description
scalar	scalar	replace old scalar with new scalar (as was done until now)
vector	scalar	replace each entry of the old vector with the same new scalar
vector	vector	works pairwise where each entry of the old vector is replaced by the corresponding entry of the new vector

In the vector-vector situation, the two vectors must have the same size.
Some examples:

1. Replacement of a scalar: only the last string `"nails"` fits the data:

```
print "Replace scalar";
scal = [ "nails" ];
old = [" lumber hammer nails "];
new = [ 1 2 3 ];
res = replace(scal,old,new);
print "Replaced scalar=",res;
```

```
Replaced scalar= 3
```

2. Illustrating single and multiple replacement on data from `assoc()` function:

```
ass0=[" lumber hammer nails , lumber hammer nails , lumber hammer   0   ,
        lumber hammer   0   , lumber   0    nails , lumber   0    nails ,
        lumber   0    nails , lumber   0    nails , lumber   0    nails ,
        lumber   0    nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer   0   ,   0    hammer   0   ,   0    hammer   0   ,
          0    hammer   0   ,   0    hammer   0   ,   0    hammer   0   ,
          0    hammer   0   ,   0    hammer   0   ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails "];
```

```
   ass1 = replace(ass0,"lumber",1);
   ass2 = replace(ass1,"hammer",2);
   ass3 = replace(ass2,"nails", 3);
   print "Single replacement: ASS3=",ass3;
```

```
Single replacement: ASS3=
   |   1   2   3
---------------
 1 |   1   2   3
 2 |   1   2   3
 3 |   1   2   0
.............
38 |   0   0   3
39 |   0   0   3
40 |   0   0   3
```

```
   old = [" lumber hammer nails "];
   new = [ 1 2 3 ];
   ass4 = replace(ass0,old,new);
   print "Multiple replacement: ASS4=",ass4;
```

We obtain the same result as with single replacement:

```
Multiple replacement: ASS4=
   |   1   2   3
---------------
```

```
 1 |   1   2   3
 2 |   1   2   3
 3 |   1   2   0
.................
38 |   0   0   3
39 |   0   0   3
40 |   0   0   3
```

3. Replace all three values by missing only:

```
   old = [" lumber hammer nails "];
   new = .;
   ass5 = replace(ass0,old,new);
   print "Multiple replacement to missing: ASS5=",ass5;
```

```
Multiple replacement to missing: ASS5=

   |         1         2         3
-----------------------------------
 1 |    .         .         .
 2 |    .         .         .
 3 |    .         .        0.00000
...................................
38 |   0.00000   0.00000    .
39 |   0.00000   0.00000    .
40 |   0.00000   0.00000    .
```

4. Large example illustrating the impovement of input:

```
  print "SAS ASSOC Data: nr=7007, nc=4";
  options NOECHO;
#include "..\\tdata\\assocs.dat"
  options ECHO;
  nr = nrow(assoc); nc = ncol(assoc);
  print "nr=",nr," nc=",nc;
```

Input for single replacement:

```
   asso = assoc[,2:4];
   ass1 = replace(asso,"apples",1);
   ass0 = replace(ass1,"artichok",2);
```

```
ass1 = replace(ass0,"avocado",3);
ass0 = replace(ass1,"baguette",4);
ass1 = replace(ass0,"bordeaux",5);
ass0 = replace(ass1,"bourbon",6);
ass1 = replace(ass0,"chicken",7);
ass0 = replace(ass1,"coke",8);
ass1 = replace(ass0,"corned_b",9);
ass0 = replace(ass1,"cracker",10);
ass1 = replace(ass0,"ham",11);
ass0 = replace(ass1,"heineken",12);
ass1 = replace(ass0,"herring",13);
ass0 = replace(ass1,"ice_crea",14);
ass1 = replace(ass0,"olives",15);
ass0 = replace(ass1,"peppers",16);
ass1 = replace(ass0,"sardines",17);
ass0 = replace(ass1,"soda",18);
ass1 = replace(ass0,"steak",19);
ass0 = replace(ass1,"turkey",20);
```

Input for multiple replacement:

```
old = ["  apples   artichok  avocado  baguette  bordeaux
          bourbon  chicken   coke     corned_b  cracker
          ham      heineken  herring  ice_crea  olives
          peppers  sardines  soda     steak     turkey "];
new = [ 1:20 ];
ass2 = replace(ass1,old,new);
```

Test for the same result:

```
ss = ssq(ass0 - ass2);
print "Error=",ss;
if (ss > .1) {
  print "Single replacement:",ass0;
  print "Multiple replacement:",ass2;
  print "Difference:",ass0 - ass2;
}
```

```
SAS ASSOC Data: nr=7007, nc=4
nr= 7007 nc= 4
Error= 0.0000
```

Note, the `replace()` function not only works for scalars, vector, and matrices, it should also work for tensors.

4.2 Extension of the `rand()` Function

A number of new uniform random generators were added:

- Mersenne-Twister (Matsumoto & Nishimura, 1998)
- AES (Advanced Encryption Standard) (Hellakalek & Wegenkittel, 2003)
- GFSR4 (Four-tap shift-register-sequence) (Ziff, 1998)
- RANLUX (Lüscher, 1994)
- Four generators using digit isolation (Richardson, 2011):
  - Twin source hexadecimal digit weave float generator
  - Twin source base 256 float generator
  - Single source base 256 integer generator with single 256 digit deck
  - Twin source base 256 integer generator with four 65536 digit decks

The syntax of the `rand()` function is unchanged:

a = rand(nr<,nc<,mtyp<,dist<,...> . >)

Below is a table of the available string options which may be used as optional input argument number four:

Distr.	Add. Arg.	Description
"icmp"	a, b	uniform RNG, very bad 16 bit version in Watcom C Compiler, int version
"iuni"	a, b	uniform with lower range a and upper range b, int version Mooore, RAND Corporation, see Fishman, p. 605
"iacm"	a, b	random generator by Schrage (1979) in ACM, int version this is not a good choice
"ikis"	a, b	random generator KISS by Marsaglia & Tsang, int version
"iecu"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), int version
"iec2"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), int version
"imwc"	a, b	multiply-with-carry RNG (Marsaglia, 2003), period 2^{128}, int version
"ix128"	a, b	XOR RNG (Marsaglia, 2003), period 2^{128}, int version
"iwow"	a, b	modified XOR RNG (Marsaglia, 2003), period $2^{192} - 2^{32}$, int version
"imet"	a, b	Mersenne-Twister (Matsumoto & Nishimura, 1998), int version
"iase"	a, b	uniform AES RNG (Hellakalek & Wegenkittel, 2003), int version
"igfs"	a, b	uniform GFSR4 RNG (Ziff, 1998), int version
"ilux"	a, b	uniform RANLUX RNG (Lüscher, 1994), int version
"itsh"	a, b	twin source hexadecimal (Richarson,2011), int version
"its7"	a, b	twin source base 256 (Richarson,2011), int version
"itss"	a, b	single source base 256 (Richarson,2011), int version
"its4"	a, b	twin source base 256 (Richarson,2011), int version

Distr.	Add. Arg.	Description
"dcmp"	a, b	uniform with lower range a and upper range b very bad 16 bit version in Watcom C Compiler, real version
"duni"	a, b	uniform with lower range a and upper range b Mooore, RAND Corporation, see Fishman, p. 605, real version
"dacm"	a, b	uniform random generator by Schrage (1979) in ACM, real version this is not a good choice
"dkis"	a, b	uniform random generator KISS by Marsaglia & Tsang, real version
"decu"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), real version
"dec2"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), real version
"dmwc"	a, b	multiply-with-carry RNG (Marsaglia, 2003), period 2^{128}, real version
"dx128"	a, b	XOR RNG (Marsaglia, 2003), period 2^{128}, real version
"dwow"	a, b	modified XOR RNG (Marsaglia, 2003), period $2^{192} - 2^{32}$, real version
"dmet"	a, b	Mersenne-Twister (Matsumoto & Nishimura, 1998), real version
"dase"	a, b	uniform AES RNG (Hellakalek & Wegenkittel, 2003), real version
"dgfs"	a, b	uniform GFSR4 RNG (Ziff, 1998), real version
"dlux"	a, b	uniform RANLUX RNG (Lüscher, 1994), real version
"dtsh"	a, b	twin source hexadecimal (Richarson,2011), real version
"dts7"	a, b	twin source base 256 (Richarson,2011), real version
"dtss"	a, b	single source base 256 (Richarson,2011), real version
"dts4"	a, b	twin source base 256 (Richarson,2011), real version

Some of the new uniform random generators were also added to the list runtime options:

Runtime Options		
Option	**Spec**	**Short Description**
RANDUNI		uniform RNG from RAND Corporation is used (SAS, IBM) (default)
RANDKISS		uniform RNG KISS by Marsaglia & Tsang (2002) (good)
RANDLECU		uniform RNG by L'Ecuyer (1999) is used (very good)
RANDLEC2		uniform RNG by L'Ecuyer (1999) is used (very good)
RANDMER		uniform Mersenne-Twister RNG (Matsumoto & Nishimura, 1998) (good)
RANDAES		uniform AES RNG (Hellakalek & Wegenkittel, 2003) (good)
RANDGFSR		uniform GFSR4 RNG (Ziff, 1998) (good)
RANDLUX		uniform RANLUX RNG (Lüscher, 1994) (good)
RANDXOR32		uniform RNG XOR32 is used, period $2^{32}-1$
RANDXOR64		uniform RNG XOR64 is used, 64bit, period $2^{64}-1$
RANDXORWOW		uniform RNG XORWOW is used, period $2^{192}-2^{32}$ (very good)
RANDXOR128		uniform RNG XOR128 is used, period $2^{128}-1$ (good)
RANDMWC3		uniform RNG MWC3 is used, period $2^{128}-1$ (good)
RANDCMP		uniform RNG of the host compiler is used (bad, period $2^{16}-1$)
RANDACM		uniform RNG by Schrage in ACM TOMS (1979), not good

Some examples:

1. Mersenne-Twister (Matsumoto & Nishimura, 1998)

```
srand(1,"imet");                          Mersenne-Twister: Int in [0,maclong]
nr = 10;
print "Mersenne-Twister: Int in [0,maclong]IMET=
me1 = rand(nr,1,'g',"imet");                 |             1
print "IMET=", me1;                       ------------------
                                           1 |    1791095845
                                           2 |    4282876139
                                           3 |    3093770124
                                           4 |    4005303368
                                           5 |    491263.000
                                           6 |     550290313
                                           7 |    1298508491
                                           8 |    4290846341
                                           9 |     630311759
                                          10 |    1013994432
```

```
print "Mersenne-Twister: Real in [0,1.]"; Mersenne-Twister: Real in [0,1.]
me2 = rand(nr,1,'g',"dmet");
print "DMET=", me2;                       DMET=
                                             |         1
                                          --------------
                                           1 |   0.09234
                                           2 |   0.39658
                                           3 |   0.18626
                                           4 |   0.38791
                                           5 |   0.34556
                                           6 |   0.66975
                                           7 |   0.39677
                                           8 |   0.93554
                                           9 |   0.53882
                                          10 |   0.84631
```

2. AES (Advanced Encryption Standard) (Hellakalek & Wegenkittel, 2003)

```
srand(1,"iaes");                          AES: Int in [0,maclong]
nr = 10;
print "AES: Int in [0,maclong]";          IAES=
aes1 = rand(nr,1,'g',"iaes");
print "IAES=", aes1;                          |             1
                                          ------------------
                                           1 |    1162513374
                                           2 |    1650716968
                                           3 |    1035058882
                                           4 |     282288633
                                           5 |     311424352
                                           6 |    2913506218
                                           7 |    2254334432
                                           8 |    3335652468
                                           9 |    2594084762
                                          10 |    3806158791
```

```
print "AES: Real in [0,1.]";
aes2 = rand(nr,1,'g',"daes");
print "DAES=", aes2;
```

```
AES: Real in [0,1.]

DAES=
   |         1
---------------
 1 |   0.56123
 2 |   0.64366
 3 |   0.49178
 4 |   0.14042
 5 |   0.93618
 6 |   0.53535
 7 |   0.07242
 8 |   0.66630
 9 |   0.37803
10 |   0.67969
```

3. GFSR4 (Four-tap shift-register-sequence) (Ziff, 1998)

```
srand(1,"igfs");
nr = 10;
print "GFSR4: Int in [0,maclong]";
gfs1 = rand(nr,1,'g',"igfs");
print "IGFSR4=", gfs1;
```

```
GFSR4: Int in [0,maclong]

IGFSR4=
   |             1
------------------
 1 |    1782013745
 2 |    2160436774
 3 |    3401042096
 4 |    1608699330
 5 |    2123337227
 6 |    2058681068
 7 |    3669225572
 8 |    1329634375
 9 |    2807004334
10 |     938964065
```

```
print "GFSR4: Real in [0,1.]";
gfs2 = rand(nr,1,'g',"dgfs");
print "DGFSR4=", gfs2;
```

```
GFSR4: Real in [0,1.]

DGFSR4=

   |         1
---------------
 1 |   0.42809
 2 |   0.29517
 3 |   0.40969
 4 |   0.29249
 5 |   0.94965
 6 |   0.63731
 7 |   0.04596
 8 |   0.52512
 9 |   0.20650
10 |   0.27707
```

4. RANLUX (Lüscher, 1994)

```
srand(1,"ilux");
nr = 10;
print "RANLUX: Int in [0,maclong]";
glux1 = rand(nr,1,'g',"ilux");
print "ILUX=", glux1;
```

```
RANLUX: Int in [0,maclong]

ILUX=
   |             1
------------------
 1 |    1702572878
 2 |     322443203
 3 |     629939111
 4 |    1721151103
 5 |    38431087.0
 6 |    1587789393
 7 |    1146929454
 8 |    1588596622
 9 |    1850774604
10 |    1095159993
```

```
print "RANLUX: Real in [0,1.]";
glux2 = rand(nr,1,'g',"dlux");
print "DLUX=", glux2;
```

```
RANLUX: Real in [0,1.]

DLUX=
   |         1
--------------
 1 |   0.16906
 2 |   0.33137
 3 |   0.66815
 4 |   0.67628
 5 |   0.33415
 6 |   0.50477
 7 |   0.13321
 8 |   0.43420
 9 |   0.44616
10 |   0.21898
```

5. Twin source hexadecimal digit weave float generator (Richardson, 2011)

```
nr = 10;
print "Richardson: Int in [0,maclong]";
print "ITSH: Seed default is 1";
srand(1,"itsh");
mu1 = rand(nr,1,'g',"itsh");
print "ITSH=", mu1;
```

```
Richardson: Int in [0,maclong]

ITSH=
   |             1
------------------
 1 |    1302985856
 2 |    1501609598
 3 |    17833190.0
 4 |    1486584758
 5 |     885442031
 6 |     140104841
 7 |     652765316
 8 |    1085543246
 9 |    1251981974
10 |    1056924350
```

```
print "Richardson: Real in [0,1]";
print "DTSH: Seed default is 1";
srand(1,"dtsh");
mu1 = rand(nr,1,'g',"dtsh");
print "DTSH=", mu1;
```

```
Richardson: Real in [0,1]

DTSH=
   |         1
---------------
 1 |   0.60675
 2 |   0.69924
 3 |   0.00830
 4 |   0.69224
 5 |   0.41232
 6 |   0.06524
 7 |   0.30397
 8 |   0.50550
 9 |   0.58300
10 |   0.49217
```

6. Twin source base 256 float generator (Richardson, 2011)

   There seems to be a bug here with that random generator:

```
print "ITS7: Seed default is 1";
srand(1,"its7");
mu2 = rand(nr,1,'g',"its7");
print "ITS7=", mu2;
```

```
ITS7=
   |             1
------------------
 1 |    1876713978
 2 |     113169975
 3 |    2122701046
 4 |    2122695781
 5 |    2122691420
 6 |    2122691440
 7 |    2122694182
 8 |    2122708144
 9 |    2122710567
10 |    2122708442
```

```
print "DTS7: Seed default is 1";          DTS7=
srand(1,"dts7");                             |         1
mu2 = rand(nr,1,'g',"dts7");                --------------
print "DTS7=", mu2;                          1 |   0.98846
                                            2 |   0.98846
                                            3 |   0.98845
                                            4 |   0.98846
                                            5 |   0.98846
                                            6 |   0.98846
                                            7 |   0.98845
                                            8 |   0.98845
                                            9 |   0.98846
                                           10 |   0.98845
```

7. Single source base 256 integer generator with single 256 digit deck (Richardson, 2011)

```
print "ISS1: Seed default is: 0xabc1234"; ISS1=
srand(1,"iss1");                             |             1
mu3 = rand(nr,1,'g',"iss1");                ------------------
print "ISS1=", mu3;                          1 |    2828909819
                                            2 |    4136048488
                                            3 |    3923840065
                                            4 |    1587099252
                                            5 |    3602636172
                                            6 |     277507532
                                            7 |    1592647447
                                            8 |     293094598
                                            9 |    3558986596
                                           10 |    2460970908

print "DSS1: Seed default is: 0xabc1234"; DSS1=
srand(1,"dss1");                             |         1
mu3 = rand(nr,1,'g',"dss1");                --------------
print "DSS1=", mu3;                          1 |   0.65866
                                            2 |   0.96300
                                            3 |   0.91359
                                            4 |   0.36953
                                            5 |   0.83880
                                            6 |   0.06461
                                            7 |   0.37082
                                            8 |   0.06824
                                            9 |   0.82864
                                           10 |   0.57299
```

8. Twin source base 256 integer generator with four 65536 digit decks (Richard-

son, 2011)

```
print "ITS4: Seed default is: 0x2fddddd3";ITS4=
srand(1,"its4");                                |             1
mu4 = rand(nr,1,'g',"its4");                   ------------------
print "ITS4=", mu4;                             1 |    1093140881
                                                2 |     903736826
                                                3 |     853341570
                                                4 |    1078002392
                                                5 |    4253615390
                                                6 |     222996434
                                                7 |    2477422813
                                                8 |    4052329872
                                                9 |    2936450153
                                               10 |    3692539166

print "DTS4: Seed default is: 0x2fddddd3";DTS4=
srand(1,"dts4");                                |          1
mu4 = rand(nr,1,'g',"dts4");                   --------------
print "DTS4=", mu4;                             1 |   0.04590
                                                2 |   0.85689
                                                3 |   0.03607
                                                4 |   0.91751
                                                5 |   0.20597
                                                6 |   0.28717
                                                7 |   0.67437
                                                8 |   0.12616
                                                9 |   0.44682
                                               10 |   0.80214
```

4.3 Extension of the `srand()` Function

The `srand()` function

srand(seed)

was extended for an optional second input argument to

srand(seed<,sopt>)

permitting the separate `seed` intialization of some uniform random generators. If the second input argument is not specified, all uniform random generators are initialized. If the second input argument is specified as one of the strings of the following table, only the corresponding random generator is reset with the new seed:

Distr.	**Add. Arg.**	**Description**
"ikis"	a, b	uniform random generator KISS by Marsaglia & Tsang, int version
"iecu"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), int version
"iec2"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), int version
"imet"	a, b	Mersenne-Twister (Matsumoto & Nishimura, 1998), int version
"iase"	a, b	uniform AES RNG (Hellakalek & Wegenkittel, 2003), int version
"igfs"	a, b	uniform GFSR4 RNG (Ziff, 1998), int version
"ilux"	a, b	uniform RANLUX RNG (Lüscher, 1994), int version
"itsh"	a, b	twin source hexadecimal (Richarson,2011), int version
"its7"	a, b	twin source base 256 (Richarson,2011), int version
"itss"	a, b	single source base 256 (Richarson,2011), int version
"its4"	a, b	twin source base 256 (Richarson,2011), int version

Distr.	Add. Arg.	Description
"dkis"	a, b	uniform random generator KISS by Marsaglia & Tsang, real version
"decu"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), real version
"dec2"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), real version
"dmet"	a, b	Mersenne-Twister (Matsumoto & Nishimura, 1998), real version
"dase"	a, b	uniform AES RNG (Hellakalek & Wegenkittel, 2003), real version
"dgfs"	a, b	uniform GFSR4 RNG (Ziff, 1998), real version
"dlux"	a, b	uniform RANLUX RNG (Lüscher, 1994), real version
"dtsh"	a, b	twin source hexadecimal (Richarson,2011), real version
"dts7"	a, b	twin source base 256 (Richarson,2011), real version
"dtss"	a, b	single source base 256 (Richarson,2011), real version
"dts4"	a, b	twin source base 256 (Richarson,2011), real version

Some examples:

1. Single source base 256 integer generator with single 256 digit deck (Richardson, 2011)

```
print "ISS1: Seed default is: 0xabc1234: hexadecimal";
srand(0xabc1234,"iss1");
mu3 = rand(nr,1,'g',"iss1");
print "ISS1=", mu3;
```

```
ISS1: Seed default is: 0xabc1234: hexadecimal

ISS1=
   |             1
------------------
 1 |     982600291
 2 |     735314717
 3 |    4087362757
 4 |    1802693449
 5 |    3799537722
 6 |     135030984
 7 |    3729099871
 8 |    2704712113
 9 |     349760511
10 |    2529659426
```

2. Twin source base 256 integer generator with four 65536 digit decks (Richardson, 2011)

```
print "ITS4: Seed default is: 0x2fddddd3: hexadecimal";
srand(0x2fddddd3,"its4");
mu4 = rand(nr,1,'g',"its4");
print "ITS4=", mu4;
```

```
ITS4: Seed default is: 0x2fddddd3: hexadecimal

ITS4=
   |             1
------------------
 1 |    3647235203
 2 |    1095945229
 3 |    3446603839
 4 |    1324252004
 5 |    3128063493
 6 |     638329403
 7 |    1752226044
 8 |    2687002194
 9 |     895754293
10 |    3862544476
```

4.4 Extension of the `mrand()` Function

The `mrand()` function generates random numbers from the following multivariate random distributions:

1. from the multivariate normal distribution $\mathcal{N}(\mu, \mathbf{\Sigma})$
2. from the multinomial distribution $\mathcal{M}(n, p_1, \ldots, p_r)$
3. uniformly distributed **inside** an n dimensional sphere with radius r
4. uniformly distributed **on** an n dimensional sphere with radius r
5. uniformly distributed **inside** an n dimensional unit cube
6. uniformly distributed **on** an n dimensional unit cube

The following multivariate distributions were added:

1. the multivariate $t(\mu, \mathbf{\Sigma}, df)$ distribution and Cauchy distribution (with default $df = 1$)
2. the multivariate Pearson distribution with Pearson parameter $c \in (-1, 1)$,
3. multivariate Khintchine distribution with parameter $df > 0$

Wang (2011) also refers to the mixed normal $\mathcal{N}(\mu_1, \mathbf{\Sigma_1}) + \mathcal{N}(\mu_2, \mathbf{\Sigma_2})$ function and includes some Matlab code by Azzalini (see Azzalini & Dalla Valle, 1996) for the multivariate skew normal distribution.
If the third argument of one of the following functions is a vector of k valid values, the `mrand()` function returns a $n \times k$ matrix of k sets of random vectors in its columns:

1. uniformly distributed **inside** an n dimensional sphere with radius r
2. uniformly distributed **on** an n dimensional sphere with radius r
3. the multivariate Pearson distribution with Pearson parameter $c \in (-1, 1)$,
4. multivariate Khintchine distribution with parameter $df > 0$

The following sparse matrix is positive definite:

```
nr = nc = 5;
rind1 = [ 1  2  3   4  4   5  5 ];
rind2 = [ 1  2  3   5      7     ];
cind  = [ 1  2  3   1  4   2  5  ];
vals  = [ 5. 5. 5.  1. 5.  2. 5. ];
a  = spmat(nr,nc,rind1,cind,vals,"coord");
c1 = (tri2sym)a;
print "A and C1", a, c1;
```

```
A and C1
 L |         1         2         3         4         5
-------------------------------------------------------
 1 |    5.0000         0         0         0         0
 2 |   0.00000    5.0000         0         0         0
 3 |   0.00000   0.00000    5.0000         0         0
 4 |   1.00000   0.00000   0.00000    5.0000         0
 5 |   0.00000    2.0000   0.00000   0.00000    5.0000


 S |         1         2         3         4         5
-------------------------------------------------------
 1 |    5.0000
 2 |   0.00000    5.0000
 3 |   0.00000   0.00000    5.0000
 4 |   1.00000   0.00000   0.00000    5.0000
 5 |   0.00000    2.0000   0.00000   0.00000    5.0000


  print "Multivariate Normal";
  srand(1);
  mu = rand(nr,1);
  print "MU=", mu, " C1=", c1;
  print "Result z is N(mu,sigma) distributed:";
  z1 = mrand("mnor",mu,c1)‘;
  print "Z1=", z1;


Multivariate Normal

MU=
   |         1
-------------
 1 |   0.18496
 2 |   0.97009
 3 |   0.39982
 4 |   0.25940
 5 |   0.92160


Z1=
   |         1         2         3         4         5
-------------------------------------------------------
 1 |   -4.7347    1.9329    2.8665   0.83277   0.25110


  print "Multivariate T and Cauchy(df=1)";
```

```
  srand(1);
  mu = rand(nr,1);
  print "MU=", mu, " C1=", c1;
  print "Result z is t(mu,sigma,df) distributed:";
  df = 2;
  z2 = mrand("tcau",mu,c1,df)‘;
  print "Z2=", z2;
```

```
Z2=
   |         1         2         3         4         5
-------------------------------------------------------
 1 |   -10.570    2.8516    2.2430    2.2367   0.20903
```

```
   srand(1);
   m = 8; c = [ .1 .3 ];
   z1 = mrand("pear",m,c);
   print "Pearson P(8,c) vectors: z1=",z1;
```

```
Pearson P(8,c) vectors: z1=
   |         1         2
------------------------
 1 |   0.13982  -0.36895
 2 |  -0.00327   -2.0098
 3 |   -1.2793  -0.00765
 4 |    1.0836    1.9431
 5 |   0.06319  -0.29222
 6 |  -0.70236   0.20316
 7 |    1.8898  -0.18800
 8 |  -0.55871  -0.08450
```

```
   srand(1);
   m = 8; df = [ 2 4 ];
   z2 = mrand("khin",m,df);
   print "Khintchine M(n=8,df) vectors: z2=",z2;
```

```
Khintchine M(n=8,df) vectors: z2=
   |         1         2
------------------------
 1 |  -0.25285   -1.6771
 2 |   0.86395   -3.5389
 3 |  -0.65445    2.0819
 4 |  -0.13342   0.86884
```

```
5 |    2.4337    1.8131
6 |   0.51949   -1.3812
7 |    0.0452    10.900
8 |    0.0192   -2.2907
```

5 New Developments

5.1 Function `assoc`

< gof,asso,nset,rules> = assoc(cust,data,<,optn<,supp>>)

Purpose: The function detects associations of items in the purchases of a set of customers. This is a typical data mining function as it is also generally available in other data mining software. The user may specify the maximum number of items in an association since the number of associations may easily rise for larger number of items n. Another way to restrict a too large number of associations are the scalar and vector specification of minimum *support*. The second output argument `asso` is needed for input in the `rules()` and `sequ()` functions. Note, this function generates results equivalent to those of PROC ASSOC and PROC RULEGEN (if either the `"minconf"` or the `"bestrul"` option is specified) in the SAS Enterprise Miner package.

Input: **cust** This must be an N vector of integers defining the customer IDs used in N purchases. There could be multiple purchases by the same customer ID. It is not necessary that the N observations of the first two input arguments are sorted w.r.t. ascending customer IDs, the `assoc()` function would do that at the input.

data This must be an $N \times n$ matrix of N purchases of n items, where `data[i,j]=1` stands for the purchase of item j by customer `cust[i]` and `data[i,j]=0` means that item j was not bought at the purchase i. This matrix is usually very sparse and maybe binary. If the data values are not binary, the `assoc()` function will transform them into binary data at the input. The binary purchase rows with the same customer ID are then combined with the union operator.

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See table below for content.

supp If this optional argument is not missing, it must be an n vector of integers defining the minimum *support* of each item at the first stage of associations. If the vector contain missing values the will be relaced by the value specified with the `"supp"` option.

Options Matrix Argument: The option argument is specified in form of a two column matrix:

Option Name	Second Column	Meaning
"bestrul"	int	number of rules with best *confidence*, *support*, or *lift*
"liftsort"		sort rules by descending *lift*
"minconf"	real	minimum *confidence* for rule generation
"minlift"	real	minimum *lift* for rule generation
"minsupp"	real	minimum *support* for rule generation
"nitem"	int	maximum number of items in an association
"pcts"	real	percentage of minimum *support* in each stage
"pini"	int	amount printed output of input data, def=0
"pass"	int	print assocoiations, def=0: no output
	1 or 3	in numeric form
	2 or 3	in string form
"prul"	int	print rules, def=0
	1 or 3	in numeric form
	2 or 3	in string form
"print"	int	amount of printed output, def=0: no output
"supp"	int	minimum *support* in each stage
"suppsort"		sort rules by descending *support*

The "minconf", "minlift", and "minsupp" specifications are combined by the logical *and* operator, that means a rule must satisfy all of them to be in the output. By default the rules are sorted by descending *confidence*, but specifying either the "liftsort" or the "suppsort" sorts them by descending *lift* or *support*.

The default value for "nitem" is $min(6, n)$. The default value for "supp" depends whether "pcts" is specified or not. If "pcts" is not specified the default of "supp" is set to 20 % of the total number of customers. For the first stage of associations the nonmissing values of the itemwise support vector specification (optional input argument 4) overrides the more general options specifications.

Output: **gof** a vector of scalar results

asso the $M \times (2+nitem)$ matrix of detected associations containing in its first column the number p of items of the association and in its second column the number of customers s where this association occurred. The remaining *nitem* columns contain the items of the association.

nset an $nitem \times 3$ matrix containing in its columns:

1. the total numbers of associations of items found,
2. the numbers of associations of items meeting the specified (or default) support level (these numbers add up to the number M of rows of the second output argument asso),
3. the maximum count number of associations with that item number.

rules only if the "minconf" or "bestrul", option is specified, returns a $M \times (7 + nitem)$ matrix of detected rules containing in its columns:

1. the number p of all items of the rule
2. the number l of left-hand side items (the number of items on the right hand side is $r = p - l$)
3. the number c_{both} of customers who selected the complete association
4. the value of *expected confidence* of the rule

$$cexp = c_{rhs}/c_{total} * 100$$

in percent where c_{rhs} the customer count of the right subset of items and c_{total} is the total number of customers of the data
5. the value of *confidence* of the rule

$$conf = c_{both}/c_{lhs} * 100$$

in percent where c_{lhs} the customer count of the left subset of items and c_{both} is the customer count of the association with both left and right item sets united
6. the percent value of *support* of the rule

$$support = c_{both}/c_{total} * 100$$

7. the *lift* value

$$lift = c_{conf}/c_{exp}$$

8. the remaining *nitem* columns contain all the p items of the rule, first those l of the left-hand side and then those $p - l$ of the right-hand side.

Restrictions: 1. Strings, missing values, and complex data are not permitted as input of the first two arguments.

2. The first two input arguments must have compatible sizes.

Relationships: sequ(), rules()

Examples: 1. Example 1: SAS/EM®Data Set $N = 40, n = 3$:

```
ass0=[" lumber hammer nails , lumber hammer nails , lumber hammer   0   ,
        lumber hammer   0   , lumber   0    nails , lumber   0    nails ,
        lumber   0    nails , lumber   0    nails , lumber   0    nails ,
        lumber   0    nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer   0   ,   0    hammer   0   ,   0    hammer   0   ,
          0    hammer   0   ,   0    hammer   0   ,   0    hammer   0   ,
          0    hammer   0   ,   0    hammer   0   ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
```

```
       0      0    nails ,   0      0    nails ,   0      0    nails ,
       0      0    nails ,   0      0    nails ,   0      0    nails ,
       0      0    nails "];


print "Data Matrix can be binary";
ass1 = replace(ass0,"lumber",1);
ass2 = replace(ass1,"hammer",1);
ass3 = replace(ass2,"nails", 1);
print "ASS3=",ass3;


optn = [ "print"    2 ,
         "pini"     2 ,
         "pass"     3 ,
         "nitem"    3 ,
         "supp"     1 ];
< gof,asso,nset,rules > = assoc(.,ass3,optn);
print "GOF=",gof;
print "Associations=",asso;
```

The missing customer argument defaults to unique ID values of $1, \ldots, N$ and the following binary data are being used:

```
      Binary Analysis Data (Rows: Customers, Columns: Items)

                  lumber hammer  nails
                1      1      1      1
                2      1      1      1
                3      1      1      0
                4      1      1      0
                5      1      0      1
                6      1      0      1
                7      1      0      1
                8      1      0      1
                9      1      0      1
               10      1      0      1
               11      0      1      1
               12      0      1      1
               13      0      1      1
               14      0      1      1
               ......................
               35      0      0      1
               36      0      0      1
               37      0      0      1
               38      0      0      1
```

39	0	0	1
40	0	0	1

Input data are sorted with unique customer number.
Total Number of Items Selected: 60
Data are stored in sparse form (60 nonzeros).

Number of Selected Items by Customers

Cust.	D.Items	Cust.	D.Items	Cust.	D.Items	Cust.	D.Items
1	3	11	2	21	1	31	1
2	3	12	2	22	1	32	1
3	2	13	2	23	1	33	1
4	2	14	2	24	1	34	1
5	2	15	2	25	1	35	1
6	2	16	2	26	1	36	1
7	2	17	2	27	1	37	1
8	2	18	2	28	1	38	1
9	2	19	1	29	1	39	1
10	2	20	1	30	1	40	1

Number of Detected Associations (Up to 3 Items)

Items	PotentSets	SuppLevSets	MaxCount
1	3	3	30
2	3	3	10
3	1	1	2
Sum	7	7	

Associations Meeting Support Level: 7
****

Dense Matrix (7 by 5)

	N_Items	Count	Item_1	Item_2	Item_3
1	1.0000000	30.000000	3.0000000	.	.
2	1.0000000	20.000000	2.0000000	.	.
3	1.0000000	10.000000	1.0000000	.	.
4	2.0000000	10.000000	2.0000000	3.0000000	.
5	2.0000000	8.0000000	1.0000000	3.0000000	.
6	2.0000000	4.0000000	1.0000000	2.0000000	.
7	3.0000000	2.0000000	1.0000000	2.0000000	3.0000000

****

```
                Associations Meeting Support Level: 7
                ************************************

                     N Count
                     1    30 nails
                     2    20 hammer
                     3    10 lumber
                     4    10 hammer nails
                     5     8 lumber nails
                     6     4 lumber hammer
                     7     2 lumber hammer nails


GOF=
              |         1
------------------------
     Failure |   0.00000
   TotalTime |   0.00000
  N_Customer |    40.000
     N_Items |    3.0000
  N_Associat |    7.0000
     N_Rules |   0.00000
  SupportLev |   1.00000
  PotentSets |    7.0000
  SupLevSets |    7.0000
      unused |   0.00000
```

2. Example 2: see Documentation of SAS Enterprise Miner:
   The following reads the data set which comes with the SAS/EM®package, version 9.1.3. It contains $N = 7007$ rows (sales/purchases) for $n = 20$ items.

```
  print "SAS ASSOC Data: nr=7007, nc=4";
  options NOECHO;
#include "..\\tdata\\assocs.dat"
  options ECHO;
  nr = nrow(assoc); nc = ncol(assoc);
  print "nr=",nr," nc=",nc;


   ass0 = assoc[,2:4];
   ass1 = replace(ass0,"apples",1);
   ass0 = replace(ass1,"artichok",2);
   ass1 = replace(ass0,"avocado",3);
   ass0 = replace(ass1,"baguette",4);
   ass1 = replace(ass0,"bordeaux",5);
```

```
ass0 = replace(ass1,"bourbon",6);
ass1 = replace(ass0,"chicken",7);
ass0 = replace(ass1,"coke",8);
ass1 = replace(ass0,"corned_b",9);
ass0 = replace(ass1,"cracker",10);
ass1 = replace(ass0,"ham",11);
ass0 = replace(ass1,"heineken",12);
ass1 = replace(ass0,"hering",13);
ass0 = replace(ass1,"ice_crea",14);
ass1 = replace(ass0,"olives",15);
ass0 = replace(ass1,"peppers",16);
ass1 = replace(ass0,"sardines",17);
ass0 = replace(ass1,"soda",18);
ass1 = replace(ass0,"steak",19);
ass0 = replace(ass1,"turkey",20);


cust = ass0[,1]; vist = ass0[,2];
nv = 20;
ind1 = [ 1 : nr ]; ind2 = ass0[,3]; ind = ind1‘ -> ind2;
/* print "Ind=", ind[1:10,]; */
ass1 = cons(nr,nv,0.);
ass1[ind] = 1;


cnam = [" apples   artichok  avocado  baguette  bordeaux
          bourbon  chicken   coke     corned_b  cracker
          ham      heineken  hering   ice_crea  olives
          peppers  sardines  soda     steak     turkey "];
ass1 = cname(ass1,cnam);
print "Sparse Binary AssoData for Input=",ass1[1:10,];
attrib(ass1); /* free ass0; */
```

The input data are binary and very sparse:

```
---------------------------------------
         Table of Attributes
---------------------------------------
Object Name      name             ass1
Object Type      otyp       matrix_gen
Data Type        dtyp              int
Storage Type     styp        spar_full
Row Names        rnam                0
Column Names     cnam               20
Row Labels       rlab                0
Column Labels    clab                0
```

```
Number Rows      nrow             7007
Number Columns   ncol               20
Lower Bandwidth  lbw              7002
Upper Bandwidth  ubw                15
Size in Bytes    size           112424
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer             7007
Smallest Value   vmin                0
Largest Value    vmax                1
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
---------------------------------------
```

We select up to 5 items with a minumum *support* of 20 as shown in the documentation of the SAS PROC ASSOC:

```
optn = [ "print"     1 ,
         "pini"      1 ,
         "pass"      3 ,
         "nitem"     5 ,
         "supp"     20 ];
< gof,asso,nset > = assoc(cust,ass1,optn);
```

```
  Input data are reduced from 7007 rows to 1001 customers.
           Total Number of Items Selected: 6938
       Data are stored in sparse form (6938 nonzeros).
```

Number of Selected Items by Customers

Cust.	D.Items	Cust.	D.Items	Cust.	D.Items	Cust.	D.Items
1	7	252	7	503	7	754	7
2	7	253	7	504	7	755	7
3	7	254	7	505	7	756	7
4	7	255	7	506	7	757	7
5	7	256	7	507	7	758	7
6	7	257	7	508	7	759	7
7	7	258	7	509	7	760	7
8	7	259	7	510	7	761	6

```
  9        7    260        6    511        7    762        7
 10        7    261        7    512        7    763        6
.....................................................................
247        7    498        7    749        7   1000        7
248        7    499        7    750        7   1001        7
249        7    500        7    751        7
250        7    501        6    752        7
251        7    502        7    753        7
```

Number of Detected Associations (Up to 5 Items)

Items	PotentSets	SuppLevSets	MaxCount
1	20	20	600
2	190	183	366
3	1035	615	234
4	1071	317	137
5	85	71	116
Sum	2401	1206	

Total Processing Time: 0

Associations Meeting Support Level: 1206

****

Dense Matrix (1207 by 7)

	N_Items	Count	Item_1	Item_2	Item_3
1	20.000000	1001.0000	.	.	.
2	1.0000000	600.00000	12.000000	.	.
3	1.0000000	488.00000	10.000000	.	.
4	1.0000000	486.00000	13.000000	.	.
5	1.0000000	473.00000	15.000000	.	.
1200	5.0000000	20.000000	7.0000000	10.000000	12.000000
1201	5.0000000	20.000000	7.0000000	11.000000	12.000000
1202	5.0000000	20.000000	8.0000000	11.000000	12.000000
1203	5.0000000	20.000000	6.0000000	9.0000000	13.000000
1204	5.0000000	20.000000	2.0000000	3.0000000	9.0000000
1205	5.0000000	20.000000	2.0000000	3.0000000	9.0000000
1206	5.0000000	20.000000	1.0000000	6.0000000	9.0000000
1207	5.0000000	20.000000	7.0000000	10.000000	12.000000

	Item_4	Item_5

```
   1 |      .          .
   2 |      .          .
   3 |      .          .
   4 |      .          .
   5 |      .          .
   ........................
1200 |   15.000000  18.000000
1201 |   14.000000  17.000000
1202 |   14.000000  17.000000
1203 |   15.000000  20.000000
1204 |   10.000000  12.000000
1205 |   11.000000  12.000000
1206 |   10.000000  16.000000
1207 |   13.000000  18.000000

     First row: number of items and number of customers


          ***************************************
          Associations Meeting Support Level: 1206
          ***************************************

        N Count
        1   600 heineken
        2   488 cracker
        3   486 herring
        4   473 olives
        5   403 bourbon
        ..............
     1200    20 chicken ham heineken ice_crea sardines
     1201    20 coke ham heineken ice_crea sardines
     1202    20 bourbon corned_b herring olives turkey
     1203    20 artichok avocado corned_b cracker heineken
     1204    20 artichok avocado corned_b ham heineken
     1205    20 apples bourbon corned_b cracker peppers
     1206    20 chicken cracker heineken herring soda


print "GOF=",gof;
print "Asso=",asso;
print "Nset=",nset;


GOF=
             |         1
-------------------------
     Failure |   0.00000
```

```
 TotalTime |   0.00000
N_Customer |    1001.0
   N_Items |    20.000
N_Associat |    1206.0
   N_Rules |   0.00000
SupportLev |    20.000
PotentSets |    2401.0
SupLevSets |    1206.0
    unused |   0.00000
```

5.2 Function `chngtxt`

b = chngtxt(a,old,new<,numb>)

Purpose: Change text in string data.

Input: **a** The first argument is a scalar, vector, matrix, or tensor object for which some of its entries should be changed.

old The second argument is either a string scalar or a vector of string values. It specifies the text with which all the entries of the first argument `a` are compared for a possible exchange.

new The third argument is either a string scalar or a vector of string values. It specifies the new text which replace(s) the old text of the input object.

rel The (optional) fourth argument specifies how many exchanges should happen. It must be either a nonnegative integer or a vector of nonnegative integers if the second input argument is a vector. Specifying zero means the number of exchanges is unrestricted. If the first input argument is a matrix or tensor, the object entries are processed rowwise. You may also specify the second argument `old` as a vector and the third argument `new` as an integer scalar. Then it relates to each of the exchanges.

Output: The only result is the modified object `b`.

Restrictions:
1. If the third argument `new` is a vector, the second argument `old` must be a vector of the same size.
2. You cannot specify the second argument as a scalar and the third argument as a vector.
3. The (optional) fourth input argument must be nonnegative integers.

Relationships: replace()

Examples:
1. Example from SAS/IML:

```
a = "It was a dark and stormy night";
b = chngtxt(a,"night","day");
print "Example from SAS/IML: b=",b;
```

```
Example from SAS/IML: b=It was a dark and stormy day
```

2. Example 1 with string matrix:

```
print "Example 1 with string matrix";
a = [" abcd  bcde cdef ,
       defg  efgh fghi "];
old = [" bcd fgh "];
new = [" wxy xyz "];
b = chngtxt(a,old,new);
print "Example with string matrix: b=",b;
```

```
Example with string matrix: b=
   |      1      2      3
-------------------------
 1 |   awxy   wxye   cdef
 2 |   defg   exyz   xyzi
```

3. Example 2 with string matrix: Restricted

```
print "Example 2 with string matrix: Restricted";
a = [" abcd  bcde cdef ,
       defg  efgh fghi "];
old = [" bcd fgh "];
new = [" wxy xyz "];
cnt = [ 1 1 ];
b = chngtxt(a,old,new,cnt);
print "Example with string matrix: b=",b;
```

```
Example with string matrix: b=
   |      1      2      3
-------------------------
 1 |   awxy   bcde   cdef
 2 |   defg   exyz   fghi
```

4. Example with mixed string tensor:

```
print "Example with mixed string tensor";
ipr = 1; srand(1);
nind  = cons(3,1,3);
a = randt(nind,"duni");


a[1,1,1] = "abcd"; a[1,1,2] = "bcde"; a[1,1,3] = "cdfg";
a[1,2,1] = "defg"; a[1,2,2] = "efgh"; a[1,2,3] = "fghi";
a[1,3,1] = "abcd"; a[1,3,2] = "bcde"; a[1,3,3] = "cdfg";
/* print "Mixed string tensor A=",a; */
a[3,1,1] = "defg"; a[3,1,2] = "efgh"; a[3,1,3] = "fghi";
```

```
a[3,2,1] = "abcd"; a[3,2,2] = "bcde"; a[3,2,3] = "cdfg";
a[3,3,1] = "defg"; a[3,3,2] = "efgh"; a[3,3,3] = "fghi";
/* print "Mixed string tensor A=",a; */


old = [" bcd efg "];
new = [" wxy xyz "];
b = chngtxt(a,old,new);
print "Example with string tensor: B=",b;
```

```
*************************
Tensor b with 3 Dimensions
*************************

          b_1
          ***

  Dense Matrix (3 by 3)

    |       1      2      3
  ------------------------
  1 |    awxy   wxye   cdfg
  2 |    dxyz   xyzh   fghi
  3 |    awxy   wxye   cdfg


          b_2
          ***

  Dense Matrix (3 by 3)

    |          1          2          3
  ---------------------------------------
  1 |   0.0665666  0.8193186  0.5238705
  2 |   0.8533943  0.0671846  0.9570239
  3 |   0.2971940  0.2726118  0.6899296


          b_3
          ***

  Dense Matrix (3 by 3)

    |       1      2      3
  ------------------------
  1 |    dxyz   xyzh   fghi
```

```
2 |    awxy   wxye   cdfg
3 |    dxyz   xyzh   fghi
```

5.3 Function `convhull`

< gof,vert,offs,cent,neigh,norm> = convhull(x<,optn<,thresh<,bounds<,feapnt> . >)

Purpose: The function computes the convex hull of a small dimensional data set x. The `qhull` package is being used (Barber, Dobkin, & Huhdanpaa, 1996) as it is used in Matlab, Octave, and the R package `geometry`. (Please, see the copyright notice.)

Input: x Should be a $npnt \times ndim$ matrix of real data, containing the coordinates of $npnt > ndim$ point in $ndim \geq 2$ dimensions, and must not contain any string, complex, or missing data.

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See table below for content.

tresh must be a $ndim \times 2$ real matrix of thresholds, default is $[-MACBIG, MACBIG]$ for missing values

bounds must be a $ndim \times 2$ real matrix of bounds, default is $[-MACBIG, MACBIG]$ for missing values

feapnt must be a $ndim$ vector specifying a feasible point which must be specified for halfspace analysis (missing values are not permitted)

Options Matrix Argument: The option argument is specified in form of a two column matrix (see the document of the `qhull` software):

Option Name	**2nd Col**	qhull	**Meaning**
`"ang"`	< 0	**A-n**	cosine of maximum angle for pre-merging
`"ang"`	> 0	**An**	cosine of maximum angle for post-merging
`"cent"`	< 0	**C-n**	centrum radius for pre-merging
`"cent"`	> 0	**Cn**	centrum radius for post-merging
`"dround"`	real	**Un**	max roundoff error for distance computation
`"farea"`	-	**FA**rea	compute total area and volume
`"fdcin"`	-	**Fd**-cdd-in	use format for input (offset first)
`"fdcout"`	-	**FD**-cdd-out	use cdd format for normals (offset first
`"gall"`	-	**Ga**ll-points	display all points as dots
`"gcent"`	-	**Gc**entrums	display centrums (2-d, 3-d)
`"ginter"`	-	Gintersections	
`"ginn"`	-	**Gi**nner	display inner planes only (2-d, 3-d)
`"gnop"`	-	**Gn**o-Planes	do not display plane
`"gout"`	-	**Go**uter	display outer planes only (2-d, 3-d)
`"gpoin"`	-	**Gp**oints	display coplanar points and vertices as radii

Option Name	2nd Col	qhull	Meaning
"gridg"	-	**Gr**idges	display ridges (3-d)
"gtran"	-	**Gt**ransparent	display transparent 3-d Delaunay triangulation
"gvert"	-	**Gv**ertices	display vertices as spheres
"gdropd"	int	**GDn**drop-dim	drop dimension n in 3-d and 4-d output
"half"	-	Halfspace	Halfspace analysis
"nopr"	-		no printout
"pakeep"	real	**PA**rea-keep	print n largest facets by area
"parea"	-	**Fa**rea	print area for each facet
"pcopl"	-	**Fc**oplanars	print coplanar points for each facet
"pcent"	-	**FC**entrums	print centrum for each facet
	-	**FC**entrums	print Voronoi vertex ("center") for each facet
"pdropl"	real	**PDk**	print facets with normal[k] ¿= n (default 0.0)
"pdropm"	real	**Pdk**	print facets with normal[k] ¡= n
"pextr"	-	**Fx** FExtremes	print extreme points of convex hull
"pfac"	-	**Ff**acets	print facets w/o ridges
"pfakeep"	real	**PF**acet-area-keep	print facets whose area is at least n
"pforc"	-	**Po**utput-forced	force output despite precision problems
"pfxr"	-	FFacets-xridge	
"pgfac"	-	**Pg**ood-facets	print good facets only (needs 'QGn' or 'QVn ')
"pgngb"	-	**PG**ood-facet-ngbs	print neighbors of good facets
"pinc"	-	**i**ncidence	print vertices incident to each facet
"pinn"	-	**Fi**nner	print inner planes for each facet
	-	**Fi**nner	print separating hyperplanes for inner, bounded Voronoi regions
"pids"	-	**FI**Ds	print ID for each facet
"pmap"	-	**FM**aple	print Maple output (2-d and 3-d)
"pmat"	-	**m**athematica	print Mathematica output (2-d and 3-d)
"pmerg"	-	**Fm**erges	print merge count for each facet (511 max)
"pmkeep"	real	**PM**erge-keep	print n facets with most merges
"pnorm"	-	**n**ormals	print hyperplane normals with offsets
"pneig"	-	**Fn**eighbors	print neighboring facets for each facet
"poff"	-	**o**ffFile	print OFF file format (dim, points and facets)
"popt"	-	**FO**ptions	print options
"pout"	-	**Fo**uter	print outer planes for each facet
	-	**Fo**uter	print separating hyperplanes for outer, unbounded Voronoi regions
"ppnti"	-	Fpoint-intersect	
"ppntn"	-	FPoint-nearest	

Option Name	2nd Col	qhull	Meaning
"ppoi"	-	**p**oints	print vertex and point coordinates
"pprec"	-	**Pp**recision	do not report precision problems
"pqhul"	-	**FQ**hull	print command for qhull and input
"print"	int		amount of printed output
"psiz"	-	**FS**ize	print sizes: total area and volume
"pstat"	-	**Ts** Pstatistics	print statistics
"psum"	-	**s**ummary	print summary
"ptria"	-	**Ft**riangles	print triangulation with added points
"pvave"	-	**FV**ertex-average	print average vertex
"pvert"	-	**Fv**ertices	print vertices for each facet
	-	**Fv**ertices	print Voronoi diagram as ridges for each input pair
"pvneig"	-	**FN**eighbors-vertex	print neighboring facets for each point
"qbbuni"	-	**QbB**ound-unit-box	scale input to fit the unit cube
"qblast"	-	**Qbb**ound-last	scale last coordinate to [0,m] for Delaunay
"qcopl"	-	**Qc**oplanar-keep	keep coplanar points with nearest facet
"qfout"	-	**Qf**urthest-outside	partition point to furthest outside facet
"qgonly"	-	**Qg**ood-facets-only	only build good facets (needs 'QGn', 'QVn ', or 'Pdk')
"qkint"	-	**Qi**nterior-keep	keep interior points with nearest facet
"qmonly"	-	**Qm**ax-outside-only	process points only if they would increase the max. outer plane
"qrand"	-	**Qr**andom-outside	process random outside points instead of furthest one
"qsimp"	-	**Qs**earch-init-simplex	search all points for the initial simplex
"qtria"	-	**Qt**riangulate	triangulated output
"qupdel"	-	**Qu**pperDelaunay	compute upper hull for furthest-site Delaunay triangulation
"qvngb"	-	**Qv**ertex-ngbs-cvx	test vertex neighbors for convexit
"qxact"	-	**Qx**act-merge	exact pre-merges (allows coplanar facets)
"qzinf"	-	**Qz**-infinity-point	add a point-at-infinity for Delaunay triangulations
"qnopre"	-	**Q0**-no-premerge	do not pre-merge facets with 'C-0' or 'Qx'
"qnoang"	-	**Q1**-no-angle-sort	sort merges by type instead of angle
"qnoarr"	-	**Q10**-no-narrow	no special processing for narrow distributions
"qtrino"	-	**Q11**-trinormals	copy normals and recompute centrums for tricoplanar facets
"qnomind"	-	**Q2**-no-merge-indep	merge all non-convex at once instead of independent sets
"qnomver"	-	**Q3**-no-merge-verts	do not merge redundant vertices

Option Name	2nd Col	qhull	Meaning
"qavoid"	-	**Q4**-avoid-old-into-new	avoid merging old facets into new facets
"qnochout"	-	**Q5**-no-check-outer	do not correct outer planes at end of qhull
"qnomconc"	-	**Q6**-no-concave-merge	do not pre-merge concave or coplanar facets
"qvirtmem"	-	**Q7**-no-breadth-first	process facets depth-first instead of breadth-first
"qnoneari"	-	**Q8**-no-near-inside	ignore near-interior points
"qpickfur"	-	**Q9**-pick-furthest	process furthest of furthest points
"ptestp"	real	QTestPoints	
"qgoodp"	< 0	**QG**ood-if-dont-see	good facet if not visible from point -n
"qgoodp"	> 0	**QG**ood-if-see-point	good facet if visible from point n
"qjogg"	real	**QJ**oggle	joggled input to avoid precision problems
"qrotat"	< 0	**QR**andom-seed	random rotation (n=-1 time/no rotate)
"qrotat"	> 0	**QR**otate-id	random rotation (n=seed, n=0 time)
"qvert"	< 0	**QV**-good-facets-not	good facet if it not includes point -n
"qvert"	> 0	**QV**-good-facets	good facet if it includes point n
"randp"	real	**Rn**	randomly perturb computations by a factor
"tcheckf"	-	**Tc**heck-frequently	check frequently during execution
"tstat"	-	**Ts**tatistics	print statistics (see "pstat")
"tveri"	-	**Tv**erify	verify result: structure, convexity, and point inclusion
"tcone"	real	**TC**one-stop	stop qhull after building cone for point n
"tfacet"	real	**TF**acet-log	report progress whenever n or more facets created
"tracep"	real	P**TPn** Trace-point	turn on tracing when point n added to hull
"tracem"	real	**TMn** Trace-merge	turn on tracing at merge n
"trerun"	real	**TR**erun	rerun qhull n times
"tvstop"	< 0	**TV**-stop-before	stop qhull before adding point n
"tvstop"	> 0	**TV**-stop-after	stop qhull after adding point n
"twidet"	real	**TW**ide-trace	trace merge facets when width ¿ n
"ucop"	real	**Un**-coplanar	max distance below plane for a new, coplanar point
"voro"	-	Voronoi	
"visi"	real	**Vn**isible	min distance above plane for a visible facet
"wout"	real	**Wn**-outside	min distance above plane for outside points

Output: **gof** column vector with some scalar results.

vert $nfac \times ndim$ matrix of the vertex points of $nfac$ facets in form of row indices of the input points x.

offs $nfac \times 3$ matrix with facet offset, inner, and facet area

cent $nfac \times ndim$ matrix of facet centers

neigh an $nfac \times nfac$ binary matrix of facet neighbors that is usually symmetric

norm $nfac \times ndim$ matrix of ther facet normals

Restrictions: 1. Currently no missing values, complex data, or strings are permitted in input argument x.

2. For compatibility of options see the `qhull` software.

Relationships: delaunay(), voronoi()

Examples: 1. Example 1: nr=7, nc=3: Qt Option

```
print "Example 1: nr=7, nc=3: Qt Option";
print "rbox c D3 | qhull_11 Qt f i n p s FA Fn Fv Fi";
ex1 = [ -0.5000      -0.5000      -0.5000,
        -0.5000      -0.5000       0.5000,
        -0.5000       0.5000      -0.5000,
        -0.5000       0.5000       0.5000,
         0.5000      -0.5000      -0.5000,
         0.5000      -0.5000       0.5000,
         0.5000       0.5000      -0.5000,
         0.5000       0.5000       0.5000 ];


print "2-column matrix options";
optn = [ "Qtria"      ,
         "pvert"      , /* v:26: vertices */
         "pfac"       , /* f: 5: print facets */
         "pinci"      , /* i:14: vertices incident to each factor */
         "pnorm"      , /* n:11: print normals with offsets */
         "ppoi"       , /* p:21: print point coordinates */
         "psum"       , /* s:24: print summary */
         "pneig"      , /* 10: print neighbors */
         "pvneig"     , /* 27: print neighbor vertex */
         "ptria"      , /* 25: print triangle */
         "pvave"      , /* 02: print vertex average */
         "pextr"      , /* 28: print extremes */
         "pstat"      ,
         "print"    4 ];
< gof,vert,offs,cent,neigh,norm > = convhull(ex1,optn);
```

```
CONVHULL() invoked by options:   Qtriangulate  PVertices  Pfacets
       Pincidence  Pnormals  Ppoints  Psummary  Pneighbors
   PNeighbors-vertex  PTriangles  PVertex-average  PExtremes
                     Pstatistics  Farea


 Analyzed Data Points: Number Points=8 Dimension=3
 -------------------------------------------------

    1|        -0.5        -0.5        -0.5
    2|        -0.5        -0.5         0.5
    3|        -0.5         0.5        -0.5
    4|        -0.5         0.5         0.5
    5|         0.5        -0.5        -0.5
    6|         0.5        -0.5         0.5
    7|         0.5         0.5        -0.5
    8|         0.5         0.5         0.5


                 Convex hull of 8 points in 3-d
                 ******************************
 Options:   Qtriangulate  Pfacets  Pincidence  Pnormals  Ppoints
                      Psummary  Pstatistics
 Number of vertices. . . . . .                          8
 Number of facets. . . . . .                           12
 Number of triangulated facets . . . . . .              6
 Number of points processed. . . . . .                  8
 Number of hyperplanes created . . . . . .             11
 Number of distance tests for qhull. . . . . .         35
 Number of distance tests for merging. . . . . .      108
 Number of distance tests for checking . . . . . .     56
 Number of merged facets . . . . . .                    6
 CPU seconds to compute hull (after input) . . . . . .  0
 Approximate facet area. . . . . .                      6
 Approximate volume. . . . . .                          1
```

The "pvave" output:

```
                          Vertex Average
                          --------------

                          0           0           0
```

The "pfac" output:

```
                     Vertices (8) and Facets (12)
```

```
                    ----------------------------

- p6 (v6):   0.5   0.5  -0.5
  neighbors: f17 f18 f25 f27 f33 f34
- p2 (v2):  -0.5   0.5  -0.5
  neighbors: f17 f29 f33
- p0 (v0):  -0.5  -0.5  -0.5
  neighbors: f17 f18 f21 f22 f29 f30
- p4 (v1):   0.5  -0.5  -0.5
  neighbors: f18 f21 f25
- p5 (v7):   0.5  -0.5   0.5
  neighbors: f21 f22 f25 f27 f37 f38
- p1 (v3):  -0.5  -0.5   0.5
  neighbors: f22 f30 f37
- p7 (v5):   0.5   0.5   0.5
  neighbors: f27 f34 f38
- p3 (v8):  -0.5   0.5   0.5
  neighbors: f29 f30 f33 f34 f37 f38


- f17
    - flags: top simplicial tricoplanar seen keepcentrum
    - area: 0.5
    - normal:            0           0          -1
    - offset:       -0.5
    - center:            0           0        -0.5
    - vertices: p6 (v6) p2 (v2) p0 (v0)
    - neighboring facets:  f29 f18 f33

- f18
    - flags: bottom simplicial tricoplanar seen
    - area: 0.5
    - normal:            0           0          -1
    - offset:       -0.5
    - center:            0           0        -0.5
    - vertices: p6 (v6) p4 (v1) p0 (v0)
    - neighboring facets:  f21 f17 f25

- f21
    - flags: top simplicial tricoplanar keepcentrum
    - area: 0.5
    - normal:            0          -1           0
    - offset:       -0.5
    - center:            0        -0.5           0
    - vertices: p5 (v7) p4 (v1) p0 (v0)
    - neighboring facets:  f18 f22 f25
```

```
- f22
    - flags: bottom simplicial tricoplanar seen
    - area: 0.5
    - normal:            0          -1           0
    - offset:       -0.5
    - center:            0        -0.5           0
    - vertices: p5 (v7) p1 (v3) p0 (v0)
    - neighboring facets:  f30 f21 f37


- f25
    - flags: top simplicial tricoplanar keepcentrum
    - area: 0.5
    - normal:            1           0           0
    - offset:       -0.5
    - center:          0.5           0           0
    - vertices: p5 (v7) p6 (v6) p4 (v1)
    - neighboring facets:  f18 f21 f27

- f27
    - flags: bottom simplicial tricoplanar seen
    - area: 0.5
    - normal:            1           0           0
    - offset:       -0.5
    - center:          0.5           0           0
    - vertices: p5 (v7) p6 (v6) p7 (v5)
    - neighboring facets:  f34 f38 f25

- f29
    - flags: bottom simplicial tricoplanar keepcentrum
    - area: 0.5
    - normal:           -1           0           0
    - offset:       -0.5
    - center:         -0.5           0           0
    - vertices: p3 (v8) p2 (v2) p0 (v0)
    - neighboring facets:  f17 f30 f33

- f30
    - flags: top simplicial tricoplanar
    - area: 0.5
    - normal:           -1           0           0
    - offset:       -0.5
    - center:         -0.5           0           0
    - vertices: p3 (v8) p1 (v3) p0 (v0)
    - neighboring facets:  f22 f29 f37
```

```
- f33
    - flags: bottom simplicial tricoplanar keepcentrum
    - area: 0.5
    - normal:            0           1           0
    - offset:       -0.5
    - center:            0         0.5           0
    - vertices: p3 (v8) p6 (v6) p2 (v2)
    - neighboring facets:  f17 f29 f34

- f34
    - flags: top simplicial tricoplanar
    - area: 0.5
    - normal:            0           1           0
    - offset:       -0.5
    - center:            0         0.5           0
    - vertices: p3 (v8) p6 (v6) p7 (v5)
    - neighboring facets:  f27 f38 f33

- f37
    - flags: top simplicial tricoplanar keepcentrum
    - area: 0.5
    - normal:            0           0           1
    - offset:       -0.5
    - center:            0           0         0.5
    - vertices: p3 (v8) p5 (v7) p1 (v3)
    - neighboring facets:  f22 f30 f38

- f38
    - flags: bottom simplicial tricoplanar
    - area: 0.5
    - normal:            0           0           1
    - offset:       -0.5
    - center:            0           0         0.5
    - vertices: p3 (v8) p5 (v7) p7 (v5)
    - neighboring facets:  f27 f34 f37
```

The "pneig" output:

```
Neighbors of 12 Facets
----------------------

   1 3 :   6   1   8
   2 3 :   2   0   4
   3 3 :   1   3   4
```

```
 4 3 :   7   2  10
 5 3 :   1   2   5
 6 3 :   9  11   4
 7 3 :   0   7   8
 8 3 :   3   6  10
 9 3 :   0   6   9
10 3 :   5  11   8
11 3 :   3   7  11
12 3 :   5   9  10
```

The "pnorm" output:

```
                Facet Normals and Offsets
                -------------------------

              Hull Dimension=3 Number Facets=12
              ---------------------------------

 1          0          0         -1       -0.5
 2          0          0         -1       -0.5
 3          0         -1          0       -0.5
 4          0         -1          0       -0.5
 5          1          0          0       -0.5
 6          1          0          0       -0.5
 7         -1          0          0       -0.5
 8         -1          0          0       -0.5
 9          0          1          0       -0.5
10          0          1          0       -0.5
11          0          0          1       -0.5
12          0          0          1       -0.5
```

The "pinci" output (similar "pvert"):

```
            Facet Incidences: Number of Facets=12
            -------------------------------------

                1 3 : 6 2 0
                2 3 : 4 6 0
                3 3 : 5 4 0
                4 3 : 1 5 0
                5 3 : 5 6 4
                6 3 : 6 5 7
                7 3 : 2 3 0
                8 3 : 3 1 0
                9 3 : 6 3 2
```

```
10 3 : 3 6 7
11 3 : 3 5 1
12 3 : 5 3 7
```

The "ppoi" output:

```
Input Points: Number Points=8 Dimension=3
-----------------------------------------


 1        -0.5        -0.5        -0.5
 2        -0.5        -0.5         0.5
 3        -0.5         0.5        -0.5
 4        -0.5         0.5         0.5
 5         0.5        -0.5        -0.5
 6         0.5        -0.5         0.5
 7         0.5         0.5        -0.5
 8         0.5         0.5         0.5
```

The "ptria" output:

```
Triangles: 8 Points 12 Facets 18 Ridges
---------------------------------------

        -0.5        -0.5        -0.5
        -0.5        -0.5         0.5
        -0.5         0.5        -0.5
```

The "pvert" output:

```
Facet Vertices: Number of Facets=12
-----------------------------------


 1 3 : 6 2 0
 2 3 : 6 4 0
 3 3 : 5 4 0
 4 3 : 5 1 0
 5 3 : 5 6 4
 6 3 : 5 6 7
 7 3 : 3 2 0
 8 3 : 3 1 0
 9 3 : 3 6 2
10 3 : 3 6 7
11 3 : 3 5 1
12 3 : 3 5 7
```

The "pvneig" output:

```
Vertex Neighbors: Number Points 8
---------------------------------

    6 :   7   3   2   1   0   6
    3 :  10   3   7
    3 :   8   0   6
    6 :  11   9   8   6   7  10
    3 :   4   1   2
    6 :  11   5   4   2   3  10
    6 :   9   5   4   1   0   8
    3 :  11   5   9
```

The "pextr" output:

```
Vertex Extremes: Number of Points 8
-----------------------------------

          0 1 2 3 4 5 6 7


              QHULL invoked by options:

 Qtriangulate  PVertices  Pfacets  Pincidence  Pnormals  Ppoints
Psummary  Pneighbors  PNeighbors-vertex  PTriangles
PVertex-average  PExtremes  Pstatistics  Farea  _pre-merge
_zero-centrum  Fvertices  _max-width  1  Error-roundoff 3.5e-016
_one-merge 2.4e-015  _near-inside 1.2e-014  Visible-distance
6.9e-016  U-coplanar-distance 6.9e-016  Width-outside 1.4e-015
_wide-facet 4.2e-015


                  Precision constants:

     0.5 max. abs. coordinate in the (transformed) input ('Qbd:n')
3.5e-016 max. roundoff error for distance computation ('En')
3.4e-016 max. roundoff error for angle computations
1.4e-015 min. distance for outside points ('Wn')
6.9e-016 min. distance for visible facets ('Vn')
6.9e-016 max. distance for coplanar facets ('Un')
4.2e-015 max. facet width for recomputing centrum and area
1.2e-014 max. distance for near-inside points
       1 max. cosine for pre-merge angle
6.9e-016 radius of pre-merge centrum
       1 max. cosine for post-merge angle
2.4e-015 max. distance for merging two simplicial facets
```

```
1.1e-016 max. roundoff error for arithmetic operations
1.1e-308 min. denominator for divisions zero diagonal for Gauss:
 4.44e-015 8.88e-015 1.33e-014
```

The "pstat" output:

```
                      Memory statistics:

     154 quick allocations
     181 short allocations
       4 long allocations
     256 short frees
       4 long frees
    4760 bytes of short memory in use
    6128 bytes of short memory in freelists
     424 bytes of long memory allocated (except for input)
       0 bytes of long memory in use (in 0 pieces)
   65536 bytes per memory buffer (initially 131072 bytes)
      13 calls to qh_setlarger
     6.4 average copy size


                    Freelists (bytes->count):
 16->13 24->48 32->9 48->9 176->23


  Size in Bytes: merge 24 ridge 28 vertex 44 facet 176 normal 24
         Ridge Vertices 16 Facet Vertices or Neighbors 20


  print "GOF=",gof;
  print "Vertices=",vert;
  print "Offset=",offs;
  print "Centers=",cent;
  print "Neighbors=",neigh;
  print "Normals=",norm;


GOF=
            |         1
------------------------
    Failure |   0.00000
       Time |   0.00000
    NFacets |    12.000
  NVertices |    8.0000
       Area |    6.0000
     Volume |   1.00000
```

```
     unused |     .
            |     .
            |     .
            |     .
```

Vertices=

	Vertex0	Vertex1	Vertex2
Fac_00	6	2	0
Fac_01	6	4	0
Fac_02	5	4	0
Fac_03	5	1	0
Fac_04	5	6	4
Fac_05	5	6	7
Fac_06	3	2	0
Fac_07	3	1	0
Fac_08	3	6	2
Fac_09	3	6	7
Fac_10	3	5	1
Fac_11	3	5	7

Offset=

	Offset	Inner	Area
Fac_00	-0.50000	-3e-016	0.50000
Fac_01	-0.50000	-3e-016	0.50000
Fac_02	-0.50000	-3e-016	0.50000
Fac_03	-0.50000	-3e-016	0.50000
Fac_04	-0.50000	-3e-016	0.50000
Fac_05	-0.50000	-3e-016	0.50000
Fac_06	-0.50000	-3e-016	0.50000
Fac_07	-0.50000	-3e-016	0.50000
Fac_08	-0.50000	-3e-016	0.50000
Fac_09	-0.50000	-3e-016	0.50000
Fac_10	-0.50000	-3e-016	0.50000
Fac_11	-0.50000	-3e-016	0.50000

Centers=

	DIM_1	DIM_2	DIM_3
Fac_00	0.00000	0.00000	-0.50000
Fac_01	0.00000	0.00000	-0.50000
Fac_02	0.00000	-0.50000	0.00000
Fac_03	0.00000	-0.50000	0.00000

Fac_04	0.50000	0.00000	0.00000
Fac_05	0.50000	0.00000	0.00000
Fac_06	-0.50000	0.00000	0.00000
Fac_07	-0.50000	0.00000	0.00000
Fac_08	0.00000	0.50000	0.00000
Fac_09	0.00000	0.50000	0.00000
Fac_10	0.00000	0.00000	0.50000
Fac_11	0.00000	0.00000	0.50000

Neighbors=

SYM	Fac_00	Fac_01	Fac_02	Fac_03	Fac_04	Fac_05
Fac_00	0					
Fac_01	1	0				
Fac_02	0	1	0			
Fac_03	0	0	1	0		
Fac_04	0	1	1	0	0	
Fac_05	0	0	0	0	1	0
Fac_06	1	0	0	0	0	0
Fac_07	0	0	0	1	0	0
Fac_08	1	0	0	0	0	0
Fac_09	0	0	0	0	0	1
Fac_10	0	0	0	1	0	0
Fac_11	0	0	0	0	0	1

SYM	Fac_06	Fac_07	Fac_08	Fac_09	Fac_10	Fac_11
Fac_06	0					
Fac_07	1	0				
Fac_08	1	0	0			
Fac_09	0	0	1	0		
Fac_10	0	1	0	0	0	
Fac_11	0	0	0	1	1	0

Normals=

	DIM_1	DIM_2	DIM_3
Fac_00	0.00000	0.00000	-1.00000
Fac_01	0.00000	0.00000	-1.00000
Fac_02	0.00000	-1.00000	0.00000
Fac_03	0.00000	-1.00000	0.00000
Fac_04	1.00000	0.00000	0.00000
Fac_05	1.00000	0.00000	0.00000
Fac_06	-1.00000	0.00000	0.00000

```
Fac_07 |  -1.00000   0.00000   0.00000
Fac_08 |   0.00000   1.00000   0.00000
Fac_09 |   0.00000   1.00000   0.00000
Fac_10 |   0.00000   0.00000   1.00000
Fac_11 |   0.00000   0.00000   1.00000
```

5.4 Function `delaunay`

< gof,vert,offs,cent,neigh> = delaunay(x<,optn<,thresh<,bounds> . >)

Purpose: The function computes the Delaunay triangulation a small dimensional data set x. The `qhull` package is being used (Barber, Dobkin, & Huhdanpaa, 1996) as it is used in Matlab, Octave, and the R package `geometry`. (Please, see the copyright notice.)

Input: **x** Should be a $npnt \times ndim$ matrix of real data, containing the coordinates of $npnt > ndim$ point in $ndim \geq 2$ dimensions, and must not contain any string, complex, or missing data.

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See the large table given for the convhull() function for content.

tresh must be a $ndim \times 2$ real matrix of thresholds, default is $[-MACBIG, MACBIG]$ for missing values

bounds must be a $ndim \times 2$ real matrix of bounds, default is $[-MACBIG, MACBIG]$ for missing values

Options Matrix Argument: The option argument is specified in form of a two column matrix:

Output: **gof** column vector with some scalar results.

vert $nfac \times ndim$ matrix of the vertex points of $nfac$ facets in form of row indices of the input points x.

offs $nfac \times 3$ matrix with facet offset, inner, and facet area

cent $nfac \times ndim$ matrix of facet centers

neigh an $nfac \times K$ matrix with indices of facet neighbors

Restrictions: 1. Currently no missing values, complex data, or strings are permitted in input argument x.

2. For compatibility of options see the `qhull` software.

Relationships: convhull(), voronoi()

Examples: 1. Example 1: nr=9, nc=3: Qz Option

```
print "Example 1: nr=9, nc=3: Qz Option";
print "rbox c D3 | qdelau_11 Qz f i p s FA Fn Fv";
ex1 = [ -0.5000      -0.5000      -0.5000,
        -0.5000      -0.5000       0.5000,
        -0.5000       0.5000      -0.5000,
```

```
         -0.5000       0.5000       0.5000,
          0.5000      -0.5000      -0.5000,
          0.5000      -0.5000       0.5000,
          0.5000       0.5000      -0.5000,
          0.5000       0.5000       0.5000 ];


print "2-column matrix options";
optn = [ "qzinf"      , /* Qz option: add point at infinity */
         "pfac"       , /* f: 5: print facets */
         "pinci"      , /* i:14: vertices incident to each factor */
         "pnorm"      , /* n:11: print normals with offsets */
         "ppoi"       , /* p:21: print point coordinates */
         "psum"       , /* s:24: print summary */
         "pneig"      , /* 10: print neighbors */
         "pvneig"     , /* 27: print neighbor vertex */
         "ptria"      , /* 25: print triangle */
         "pvave"      , /* 02: print vertex average */
         "pextr"      , /* 28: print extremes */
         "pstat"      ,
         "print"    4 ];
< gof,vert,offs,cent,neigh > = delaunay(ex1,optn);


 DELAUNAY() invoked by options:   Qz-infinity-point  Pfacets
     Pincidence  Pnormals  Ppoints  Psummary  Pneighbors
  PNeighbors-vertex  PTriangles  PVertex-average  PExtremes
     Pstatistics  QbBound-unit-box  Qcoplanar-keep  Farea


Analyzed Data Points: Number Points=9 Dimension=4
-------------------------------------------------

  1|        -0.5        -0.5        -0.5        0.75
  2|        -0.5        -0.5         0.5        0.75
  3|        -0.5         0.5        -0.5        0.75
  4|        -0.5         0.5         0.5        0.75
  5|         0.5        -0.5        -0.5        0.75
  6|         0.5        -0.5         0.5        0.75
  7|         0.5         0.5        -0.5        0.75
  8|         0.5         0.5         0.5        0.75
  9|           0           0           0       0.825


 Delaunay triangulation by the convex hull of 9 points in 4-d:
 ************************************************************
 Options:   Qz-infinity-point  Pfacets  Pincidence  Pnormals
```

```
                      Ppoints  Psummary
                         Pstatistics
 Number of input sites and at-infinity . . . . . .          9
 Number of Delaunay regions. . . . . .          1
 Number of non-simplicial Delaunay regions . . . . . .          1
 Number of points processed. . . . . .          9
 Number of hyperplanes created . . . . . .         12
 Number of facets in hull. . . . . .          7
 Number of distance tests for qhull. . . . . .         45
 Number of distance tests for merging. . . . . .        147
 Number of distance tests for checking . . . . . .         72
 Number of merged facets . . . . . .         10
 CPU seconds to compute hull (after input) . . . . . .          0
 Approximate facet area. . . . . .          1
```

The "pvave" output:

```
                        Vertex Average
                        --------------


                    0           0           0           0
```

The "pfac" output:

```
                    Vertices (9) and Facets (7)
                    ---------------------------

- p3 (v9):  -0.5   0.5   0.5     0
  neighbors: f1 f4 f13 f17
- p5 (v8):   0.5  -0.5   0.5     0
  neighbors: f1 f3 f12 f17
- p6 (v7):   0.5   0.5  -0.5     0
  neighbors: f1 f2 f12 f13
- p7 (v6):   0.5   0.5   0.5     0
  neighbors: f1 f13 f12 f17
- p1 (v3):  -0.5  -0.5   0.5     0
  neighbors: f1 f3 f4 f17
- p2 (v2):  -0.5   0.5  -0.5     0
  neighbors: f1 f2 f4 f13
- p4 (v1):   0.5  -0.5  -0.5     0
  neighbors: f1 f2 f3 f12
- p0 (v0):  -0.5  -0.5  -0.5     0
  neighbors: f1 f2 f3 f4
```

```
- f1
    - flags: top tested seen coplanar
    - area:  1
    - merges: 4
    - normal:            0           0           0          -1
    - offset:          0
    - center:            0           0           0           0
    - vertices: p3 (v9) p5 (v8) p6 (v7) p7 (v6) p1 (v3) p2 (v2)
                p4 (v1) p0 (v0)
    - neighboring facets:  f2 f3 f4 f17 f12 f13
    - ridges:
     - r4 tested
           vertices: p2 (v2) p4 (v1) p0 (v0)
           between f1 and f2
     - r7 tested
           vertices: p6 (v7) p2 (v2) p4 (v1)
           between f2 and f1
     - r5 tested
           vertices: p1 (v3) p4 (v1) p0 (v0)
           between f3 and f1
     - r15 tested
           vertices: p5 (v8) p1 (v3) p4 (v1)
           between f1 and f3
     - r6 tested
           vertices: p1 (v3) p2 (v2) p0 (v0)
           between f1 and f4
     - r25 tested
           vertices: p3 (v9) p1 (v3) p2 (v2)
           between f4 and f1
     - r17 tested
           vertices: p5 (v8) p7 (v6) p1 (v3)
           between f1 and f17
     - r27 tested
           vertices: p3 (v9) p7 (v6) p1 (v3)
           between f17 and f1
     - r8 tested
           vertices: p6 (v7) p7 (v6) p4 (v1)
           between f1 and f12
     - r16 tested
           vertices: p5 (v8) p7 (v6) p4 (v1)
           between f12 and f1
     - r9 tested
           vertices: p6 (v7) p7 (v6) p2 (v2)
           between f13 and f1
     - r26 tested
           vertices: p3 (v9) p7 (v6) p2 (v2)
```

```
between f1 and f13
```

The "pneig" output:

```
                    Neighbors of 1 Facets
                    ---------------------

                  1 6 :  -2  -3  -4 -17 -12 -13
```

The "pnorm" output:

```
                   Facet Normals and Offsets
                   -------------------------

                Hull Dimension=4 Number Facets=1
                --------------------------------

   1           0           0           0          -1           0
```

The "pinci" output (similar "pvert"):

```
              Facet Incidences: Number of Facets=12
              -------------------------------------

                            1 10 : 1 2 0
                            2 10 : 2 4 0
                            3 10 : 4 1 0
                            4 10 : 5 7 1
                            5 10 : 7 6 2
                            6 10 : 2 6 4
                            7 10 : 6 7 4
                            8 10 : 5 1 4
                            9 10 : 7 5 4
                           10 10 : 1 3 2
                           11 10 : 3 7 2
                           12 10 : 7 3 1
```

The "ppoi" output:

```
             Input Points: Number Points=9 Dimension=4
             -----------------------------------------


         1        -0.5        -0.5        -0.5           0
         2        -0.5        -0.5         0.5           0
```

```
3        -0.5         0.5        -0.5          0
4        -0.5         0.5         0.5          0
5         0.5        -0.5        -0.5          0
6         0.5        -0.5         0.5          0
7         0.5         0.5        -0.5          0
8         0.5         0.5         0.5          0
```

The "ptria" output:

```
Triangles: 10 Points 12 Facets 3 Ridges
---------------------------------------

          -0.5        -0.5        -0.5
          -0.5        -0.5         0.5
          -0.5         0.5        -0.5

             Point Centers
             -------------


1            0           0           0
```

The "pvneig" output:

```
 Vertex Neighbors: Number Points 9
 ---------------------------------

         4 :  -4  -3  -2   0
         4 : -17  -4  -3   0
         4 : -13  -4  -2   0
         4 : -17 -13  -4   0
         4 : -12  -3  -2   0
         4 : -17 -12  -3   0
         4 : -13 -12  -2   0
         4 : -17 -13 -12   0
         0 :
```

The "pextr" output:

```
Vertex Extremes: Number of Points 8
-----------------------------------

          3 5 6 7 1 2 4 0
```

```
                    QHULL invoked by options:

 Qz-infinity-point  Pfacets  Pincidence  Pnormals  Ppoints
Psummary  Pneighbors  PNeighbors-vertex  PTriangles
PVertex-average  PExtremes  Pstatistics  QbBound-unit-box
Qcoplanar-keep  Farea  _pre-merge  _zero-centrum  Pgood
Qinterior-keep  _max-width  1  Error-roundoff 1e-015  _one-merge
9.1e-015  Visible-distance 6e-015  U-coplanar-distance 6e-015
Width-outside 1.2e-014  _wide-facet 3.6e-014

                    Precision constants:

      1 max. abs. coordinate in the (transformed) input ('Qbd:n')
  1e-015 max. roundoff error for distance computation ('En')
4.5e-016 max. roundoff error for angle computations
1.2e-014 min. distance for outside points ('Wn')
  6e-015 min. distance for visible facets ('Vn')
  6e-015 max. distance for coplanar facets ('Un')
3.6e-014 max. facet width for recomputing centrum and area
      1 max. cosine for pre-merge angle
  2e-015 radius of pre-merge centrum
      1 max. cosine for post-merge angle
9.1e-015 max. distance for merging two simplicial facets
1.1e-016 max. roundoff error for arithmetic operations
2.2e-308 min. denominator for divisions zero diagonal for Gauss:
 4.44e-015 8.88e-015 1.33e-014 2.22e-014
```

The "pstat" output:

```
                     Memory statistics:

    153 quick allocations
    161 short allocations
      5 long allocations
    191 short frees
      5 long frees
   6192 bytes of short memory in use
   2200 bytes of short memory in freelists
    424 bytes of long memory allocated (except for input)
      0 bytes of long memory in use (in 0 pieces)
  65536 bytes per memory buffer (initially 131072 bytes)
     27 calls to qh_setlarger
    5.2 average copy size

                   Freelists (bytes->count):
```

```
 24->25 32->3 48->1 176->8


  Size in Bytes: merge 24 ridge 28 vertex 44 facet 176 normal 32
         Ridge Vertices 20 Facet Vertices or Neighbors 24


  print "GOF=",gof;
  print "Vertices=",vert;
  print "Offset=",offs;
  print "Centers=",cent;
  print "Neighbors=",neigh;


GOF=
            |         1
------------------------
    Failure |   0.00000
       Time |   0.00000
    NFacets |    7.0000
  NVertices |    9.0000
       Area |   1.00000
     Volume |   0.00000
     unused |    .
            |    .
            |    .
            |    .


Vertices=
          |   Vertex0   Vertex1   Vertex2   Vertex3
-----------------------------------------------------
 Fac_00   |         3         5         6         7

          |   Vertex4   Vertex5   Vertex6   Vertex7
-----------------------------------------------------
 Fac_00   |         1         2         4         0


Offset=
          |    Offset     Inner      Area
-----------------------------------------
 Fac_00   |   0.00000   -1e-015   1.00000


Centers=
     ZER |   DIM_1   DIM_2   DIM_3   DIM_4
-----------------------------------------
```

Fac_00	0	0	0	0	

Neighbors=

	Fac_0	Fac_1	Fac_2	Fac_3	Fac_4	Fac_5
Fac_0	-2	-3	-4	-17	-12	-13

5.5 Function cperm

B = cperm(A<,perm>)

Purpose: Random or specified permutation of the N columns of a matrix. This can also be done more generally by using index vectors but maybe this more specific implementation could be faster.

Input: **A** specifies an $N \times n$ matrix.

perm (optional) an n vector of integers defining the permutation applied to the n columns of matrix **A**. If `perm` is missing, it is defined by a random permutation. (Use the `srand()` for specifying a random seed.)

Output: The only output argument is the $N \times n$ matrix **B** containing the permuted columns of **A**.

Restrictions:

Relationships: rperm()

Examples:

```
print "[6] Mixed Type General Rectangular Matrix";
perm = [ 3 2 1 ];
rnam = [" row1:row3 "];
cnam = [" col1:col3 "];


amat = [ "a" 1 "c" ,
          2 "b" 2  ,
         "d" 3 "e" ];


amat = rname(amat,rnam); amat = cname(amat,cnam);
bmat = cperm(amat,perm);
print "[7] CPERM=",bmat;
```

```
                              bmat
                              ****

                Mixed Type Matrix bmat

                     col3 col2 col1
                row1    c    1    a
                row2    2    b    2
                row3    e    3    d
```

The result of this index operation should be the same:

```
cmat = amat[,perm];
print "[7] ColIndex=",cmat;
```

```
                              cmat
                              ****

                    Mixed Type Matrix cmat

                          col3 col2 col1
                    row1    c    1    a
                    row2    2    b    2
                    row3    e    3    d
```

5.6 Function `hankel`

B = hankel(a<,p>)

Purpose: Create a $k \times k$ Hankel matrix. There are three different situations:

- a is n vector: Result **B** is $p \times p$ symmetric matrix, maybe filled with zeros.
  - If p is specified: $k = p$:
  - If p is not specified: n must be odd integer: $n = 2 * k + 1$, $k = (int)(n/2)$.
- a is $n \times m$ matrix: There must be $max(n, m) == i * min(n, m)$ where i is integer. Result **B** is $p \times p$ unsymmetric matrix, maybe filled with zeros.
  - If p is specified: $k = p$:
  - If p is not specified: $k = max(n, m)$

Input: The first input argument must be either a n vector or a $n \times m$ matrix of numerical values (int, real, or complex). Currently, a should not contain string data.

Output: The function returns a $k \times k$ Hankel matrix with the same data type as the input vector or matrix a.

Restrictions: 1. A missing value for v is returned if the input a contains string data.

2. If a is n vector and p is not specified, n must be odd integer.
3. If a is $n \times m$ matrix there must be $max(n, m) == i * min(n, m)$ where i is integer.

Relationships:

Examples: 1. Example from Horn and Johnson (1996, p.27):

```
print "Examples from Horn and Johnson (1996, p.27)";
av1 = [ 0:10 ];
print "Hankel: B01=",b01 = hankel(av1,6);
print "Hankel: B02=",b02 = hankel(av1);
```

```
Example from Horn and Johnson

Hankel: B01=
 S |    1    2    3    4    5    6
```

```
---------------------------------
 1 |    0
 2 |    1    2
 3 |    2    3    4
 4 |    3    4    5    6
 5 |    4    5    6    7    8
 6 |    5    6    7    8    9   10


Hankel: B02=
 S |    1    2    3    4    5    6
---------------------------------
 1 |    0
 2 |    1    2
 3 |    2    3    4
 4 |    3    4    5    6
 5 |    4    5    6    7    8
 6 |    5    6    7    8    9   10
```

2. Example 1 from SAS/IML Manual:

```
print "Example 1 from IML Manual";
av2 = [ 1:5 ];
print "Hankel: B03=",b03 = hankel(av2,5);
```

```
Example 1 from IML Manual

Hankel: B03=
 S |   1   2   3   4   5
------------------------
 1 |   1
 2 |   2   3
 3 |   3   4   5
 4 |   4   5   0   0
 5 |   5   0   0   0   0
```

3. Example 2 from SAS/IML Manual:

```
print "Example 2 from IML Manual: nrow = 2*ncol";
am1 = [ 1 2, 3 4, 5 6, 7 8 ];
print "Hankel: B2=", b2 = hankel(am1);
```

```
Example 2 from IML Manual: nrow = 2*ncol
```

```
Hankel: B2=
   |   1   2   3   4
-------------------
 1 |   1   2   5   6
 2 |   3   4   7   8
 3 |   5   6   0   0
 4 |   7   8   0   0
```

4. Example 3 from SAS/IML Manual:

```
print "Example 3 from IML Manual: ncol = 2*nrow";
am2 = [ 1 2 3 4 , 5 6 7 8 ];
print "Hankel: B3=", b3 = hankel(am2);
```

```
Example 3 from IML Manual: ncol = 2*nrow

Hankel: B3=
   |   1   2   3   4
-------------------
 1 |   1   2   3   4
 2 |   5   6   7   8
 3 |   3   4   0   0
 4 |   7   8   0   0
```

5.7 Function `mvntest`

<prob,stat> = mvntest(sopt,x<,optn>)

Purpose: The `mvntest` function implements various tests for multivariate normality (Wang, 2011):

- Mardia's tests for multivariate skewness
- Mardia's tests for multivariate kurtosis
- Mardia & Foster (1983) omnibus test for skewness and kurtosis
- Royston (1983) W test (extension of Shapiro-Wilks test)
- $W_{min}(5)$ test by Wang and Hwang (2011)
- Henze-Zirkler (1990) test
- Small's $Q1$, $Q2$, $Q3$
- Doornik-Hansen Omnibus test
- Mudholkar-McDermott-Srivastava (1992) test
- Szekely-Rizzo test (2005)

The code underlying most of these tests is based on Matlab code written by C.C. Wang (2011), Trujillo-Ortiz et al. (2007), and others.

Note that for univariate normality testing CMAT provides the following tests:

- Anderson-Darling:
  p = `adtest(y<,"vers">)`
- Berkowitz:
  < `prob,stat` > = `berkow(y<,optn<,"dist"<,...>.>)`
- Jarque-Bera:
  < `prob,stat` > = `jarbera(y<,optn>)`
- Kolmogorov-Smirnov:
  < `prob,stat,hypo` > = `kstest(y<,optn<,"dist"<,...>.>)`
- Shapiro-Wilks:
  < `prob,stat` > = `shapwilk(y<,optn>)`

Input: **sopt** specifies the kind of multivariate normality test:

"mark" Mardia's test for multivariate kurtosis

"mars" Mardia's test for multivariate skewness

"maro" Mardia & Foster (1983) omnibus test for multivariate skewness and kurtosis

"wmin" $W_{min}(5)$ test by Wang and Hwang (2011)

"roys" Royston (1983) W test (MV extension of Shapiro-Wilks test)

"hezi" Henze-Zirkler (1990) test

"q123" Small's $Q1$, $Q2$, $Q3$

"doha" Doornik-Hansen Omnibus test

"mudh" Mudholkar-McDermott-Srivastava (1992) test

"szek" Szekely-Rizzo (2005) test

x Should be a $N \times p$ matrix where the columns define the variates and the rows define the sample observations.

optn a vector of more general options containing:

1. the amount of printed output (default=0: no printed output)
2. probability α for significance level
3. *noint* specifying whether $N-1$ (default noint=0) or N (noint=1) is used for the variance divisor

Output: prob a vector with the probabilities for the tests

stat a vector of the same size as `prob`, if available with the corresponding quantile of the test, otherwise with missing values

Restrictions:

Relationships: adtest(), kstest(), berkow(), jarbera(), shapwilk()

Examples: Similar to Wang (2011) the data are the first 50 observations of the four predictor variables (Sepal Length, Sepal Width, Petal Length, Petal Width) of the Iris Data. Since these data all belong to response $y = 1$ and therefore are somehow sorted, they are not expected to be very normal.

```
  print "Normality of Iris Data";
  options NOECHO;
#include "..\\tdata\\iris3.dat"
  options ECHO;
  iris = X[1:50,];
```

From the moments especially skewness and kurtosis are of interest. Note that the Matlab versions of skewness and kurtosis that are used in Wang (2011) compute the biased values (and in addition, the value of the kurtosis is not corrected by -3):

```
  sopt = [" ari var ske kur "];
  mom = univar(iris,sopt,"biask");
  print "Moments=",mom;
```

```
Moments=
           |   Sepal_Length    Sepal_Width
-----------------------------------------
  Ari_Mean |         5.0060         3.4280
  Variance |        0.12425        0.14369
  Skewness |        0.11645        0.03992
  Kurtosis |       -0.34576        0.74422

           |   Petal_Length    Petal_Width
-----------------------------------------
  Ari_Mean |         1.4620        0.24600
  Variance |        0.03016        0.01111
  Skewness |        0.10318         1.2159
  Kurtosis |        0.80459         1.4343
```

The options vector is the same for all runs except for specifying the `optn[3]=noint` option. To obtain the same results as the Matlab implementation by Wang (2011) we had to specify for all Mardia and the Henze-Zirkler algorithm `noint=1`. To obtain the same results as reported by Doornik & Hansen (1995) we must specify `noint=1` for the Doornik-Hansen method and for Mudholkar-McDermott-Srivastava `noint=0`.

```
optn = [   2 ,  /* print */
         .05 ,  /* alpha */
           1 ]; /* noint */
```

1. Results of some multivariate normal tests:
   - Mardia's tests for multivariate skewness:

     ```
     optn[3] = 1;
     < p1,s1 > = mvntest("mark",iris,optn);
     print "Mardia MV Kurtosis P:",p1;
     print "Mardia MV Kurtosis S:",s1;
     ```

     ```
       Multivariate Normality Test: Mardia MV Kurtosis Alpha= 0.05000
                 Number Observations=50 Number Variables=4
             Hypothesis H0: Data are normal distributed versus
               Hypothesis H1: Data are not normal distributed
           H0 accepted: Probability=0.0543032 Statistics=26.5377
     ```

     The result agrees with that reported by Wang (2011):

     ```
     Mardia MV Kurtosis P: 0.05430
     Mardia MV Kurtosis S: 26.538
     ```

- Mardia's tests for multivariate kurtosis

```
optn[3] = 1;
< p2,s2 > = mvntest("mars",iris,optn);
print "Mardia MV Skewness P:",p2;
print "Mardia MV Skewness S:",s2;
```

```
  Multivariate Normality Test: Mardia MV Skewness Alpha= 0.05000
            Number Observations=50 Number Variables=4
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
      H0 accepted: Probability=0.127934 Statistics=3.07972
```

The result agrees with that reported by Wang (2011):

```
Mardia MV Skewness P: 0.1279
Mardia MV Skewness S: 3.0797
```

- Mardia & Foster (1983) omnibus test for skewness and kurtosis

```
optn[3] = 1;
< p3,s3 > = mvntest("maro",iris,optn);
print "Mardia MV Omnibus P:",p3;
print "Mardia MV Omnibus S:",s3;
```

```
Multivariate Normality Test: Mardia-Foster Omnibus Alpha= 0.05000
            Number Observations=50 Number Variables=4
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
      H0 accepted: Probability=0.154825 Statistics=3.46194
```

The result agrees with that reported by Wang (2011):

```
Mardia MV Omnibus P: 0.1548
Mardia MV Omnibus S: 3.4619
```

- $W_{min}(5)$ test by Wang and Hwang (2011)

```
< p4,s4 > = mvntest("wmin",iris,optn);
print "Wang Wmin(5) P:",p4;
print "Wang Wmin(5) S:",s4;
```

```
  Multivariate Normality Test: Wang-Hwang Wmin(5) Alpha= 0.05000
            Number Observations=50 Number Variables=4
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
     H0 accepted: Probability=0.0746763 Statistics=0.941232
```

```
Wang Wmin(5) P: 0.07468
Wang Wmin(5) S: 0.9412
```

- Royston (1983) W test (MV extension of Shapiro-Wilks test)

```
optn[3] = 0;
< p5,s5 > = mvntest("roys",iris,optn);
print "Royston P:",p5;
print "Royston S:",s5;
```

```
    Multivariate Normality Test: Royston (1983) Alpha= 0.05000
            Number Observations=50 Number Variables=4
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
     H0 rejected: Probability=6.30717e-006 Statistics=29.3618
```

The result agrees with that reported by Wang (2011):

```
Royston P: 0.000006307
Royston S: 29.362
```

- Henze-Zirkler (1990) test

```
optn[3] = 1;
< p6,s6 > = mvntest("hezi",iris,optn);
print "Henze-Zirkler P:",p6;
print "Henze-Zirkler S:",s6;
```

```
  Multivariate Normality Test: Henze-Zirkler (1990) Alpha= 0.05000
              Number Observations=50 Number Variables=4
          Hypothesis H0: Data are normal distributed versus
            Hypothesis H1: Data are not normal distributed
       H0 rejected: Probability=0.0499536 Statistics=0.948845
```

The result agrees with that reported by Wang (2011):

```
Henze-Zirkler P: 0.04995
Henze-Zirkler S: 0.9488
```

- Small's $Q1$, $Q2$, $Q3$

```
optn[3] = 0;
< p7,s7 > = mvntest("q123",iris,optn);
print "Small Q123 P:",p7;
print "Small Q123 S:",s7;
```

```
         Multivariate Normality Test: Small Q1 Alpha= 0.05000
              Number Observations=50 Number Variables=4
```

```
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
      H0 rejected: Probability=0.0262917 Statistics=11.0243

       Multivariate Normality Test: Small Q2 Alpha= 0.05000
       H0 accepted: Probability=0.073763 Statistics=8.53736

       Multivariate Normality Test: Small Q3 Alpha= 0.05000
      H0 rejected: Probability=0.0121278 Statistics=19.5617
```

The result agrees with that reported by Wang (2011):

```
Small Q123 P:
          |         1
----------------------
  SmallQ1 |   0.02629
  SmallQ2 |   0.07376
  SmallQ3 |   0.01213

Small Q123 S:
          |         1
----------------------
  SmallQ1 |    11.024
  SmallQ2 |    8.5374
  SmallQ3 |    19.562
```

Doornik & Hansen (1995) call the $Q3$ omnibus measure Q_p, and report a test statistics of $Q_p = 24.0387$ for this example. The difference to the value of $Q3 = 19.562$ reported here and at Wang (2011) is probably based on different implementations for the computation of skewness and kurtosis.

- Doornik-Hansen (1995) test

```
optn[3] = 1;
< p8,s8 > = mvntest("doha",iris,optn);
print "Doornik-Hansen MV Omnibus P:",p8;
print "Doornik-Hansen MV Omnibus S:",s8;

Multivariate Normality Test: Doornik-Hansen (1995) Alpha= 0.05000
            Number Observations=50 Number Variables=4
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
      H0 rejected: Probability=0.00195219 Statistics=24.4145
```

The result agrees with that reported by Doornik & Hansen (1995):

```
Doornik-Hansen MV Omnibus P: 0.001952
```

```
Doornik-Hansen MV Omnibus S: 24.414
```

- Mudholkar-McDermott-Srivastava (1992) test

```
optn[3] = 0;
< p9,s9 > = mvntest("mudh",iris,optn);
print "Mudholkar-McDermott-Srivasta P:",p9;
print "Mudholkar-McDermott-Srivasta S:",s9;

Multivariate Normality Test: Mudholkar et.al (1992) Alpha= 0.05000
            Number Observations=50 Number Variables=4
         Hypothesis H0: Data are normal distributed versus
           Hypothesis H1: Data are not normal distributed
       H0 accepted: Probability=0.289275 Statistics=1.12299
```

The result agrees with that reported by Doornik & Hansen (1995):

```
Mudholkar-McDermott-Srivasta P: 0.2893
Mudholkar-McDermott-Srivasta S: 1.1230
```

- Szekely-Rizzo (2005) test

```
optn = [   2 ,  /* print */
         .05 ,  /* alpha */
           0 ]; /* noint */
< p10,s10 > = mvntest("szek",iris,optn);
print "Szekely and Rizzo P:",p10;
print "Szekely and Rizzo S:",s10;

 Multivariate Normality Test: Szekely-Rizzo (2005) Alpha= 0.05000
            Number Observations=50 Number Variables=4
         Hypothesis H0: Data are normal distributed versus
           Hypothesis H1: Data are not normal distributed
       H0 rejected: Probability=0.0288179 Statistics=1.2034

Szekely and Rizzo P: 0.02882
Szekely and Rizzo S: 1.2034
```

- Most of the tests in one run:

```
 sopt = [" mark mars maro wmin roys hezi q123 doha mudh szek "];
 optn = [   2 ,  /* print */
          .05 ,  /* alpha */
            1 ]; /* noint */
< prob,stat > = mvntest(sopt,iris,optn);
print "All together P:",prob;
print "All together S:",stat;
```

```
Multivariate Normality Test: Mardia MV Kurtosis Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
    H0 accepted: Probability=0.0543032 Statistics=26.5377

Multivariate Normality Test: Mardia MV Skewness Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
     H0 accepted: Probability=0.127934 Statistics=3.07972

Multivariate Normality Test: Mardia-Foster Omnibus Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
     H0 accepted: Probability=0.154825 Statistics=3.46194

Multivariate Normality Test: Wang-Hwang Wmin(5) Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
    H0 accepted: Probability=0.0811309 Statistics=0.941907

  Multivariate Normality Test: Royston (1983) Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
   H0 rejected: Probability=6.30717e-006 Statistics=29.3618

Multivariate Normality Test: Henze-Zirkler (1990) Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
    H0 rejected: Probability=0.0499536 Statistics=0.948845

Multivariate Normality Test: Small Q1,2,3 (1980) Alpha= 0.05000
          Number Observations=50 Number Variables=4
      Hypothesis H0: Data are normal distributed versus
        Hypothesis H1: Data are not normal distributed
    H0 rejected: Probability=0.0262917 Statistics=11.0243

     Multivariate Normality Test: Small Q2 Alpha= 0.05000
     H0 accepted: Probability=0.073763 Statistics=8.53736
```

```
       Multivariate Normality Test: Small Q3 Alpha= 0.05000
      H0 rejected: Probability=0.0121278 Statistics=19.5617

 Multivariate Normality Test: Doornik-Hansen (1995) Alpha= 0.05000
            Number Observations=50 Number Variables=4
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
      H0 rejected: Probability=0.00195219 Statistics=24.4145

  Multivariate Normality Test: Szekely-Rizzo (2005) Alpha= 0.05000
            Number Observations=50 Number Variables=4
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
       H0 rejected: Probability=0.0288179 Statistics=1.2034

 Multivariate Normality Test: Mudholkar et.al (1992) Alpha= 0.05000
            Number Observations=50 Number Variables=4
        Hypothesis H0: Data are normal distributed versus
          Hypothesis H1: Data are not normal distributed
       H0 accepted: Probability=0.289275 Statistics=1.12299

All together P:
                  |         1
------------------------------
   MardiaKurtosis |   0.05430
   MardiaSkewness |   0.12793
  MardFostOmnibus |   0.15482
      WangWmin(5) |   0.08113
        Royston92 |   0.00001
     HenzeZirkler |   0.04995
          SmallQ1 |   0.02629
          SmallQ2 |   0.07376
          SmallQ3 |   0.01213
    DoornikHansen |   0.00195
        Mudholkar |   0.28927
     SzekelyRizzo |   0.02882

All together S:
                  |         1
------------------------------
   MardiaKurtosis |    26.538
   MardiaSkewness |    3.0797
  MardFostOmnibus |    3.4619
      WangWmin(5) |   0.94191
        Royston92 |    29.362
```

```
     HenzeZirkler |   0.94885
          SmallQ1 |    11.024
          SmallQ2 |    8.5374
          SmallQ3 |    19.562
    DoornikHansen |    24.414
       Mudholkar |    1.1230
     SzekelyRizzo |    1.2034
```

2. Some univariate test probabilities:

   - Andersen-Darling:

```
prob = adtest(iris,"norm");
print "Anderson-Darling Norm: Prob4=",prob;
```

```
Anderson-Darling Norm: Prob4=
   |         1
--------------
 1 |   0.33524
 2 |   0.21018
 3 |   0.01079
 4 |    7e-012
```

   - Jarque-Bera:

```
print "Jarque-Bera test for Normality: nvar=1";
optn = [   2 ,  /* print */
         .05 ,  /* alpha */
           2 ]; /* nvar */
< p1,s1 > = jarbera(iris,optn);
print "JARBERA P:",p1;
print "JARBERA S:",s1;
```

```
            Jarque-Bera Test for Normal Distribution
          Hypothesis H0: Data are normally distributed versus
           Hypothesis H1: Data are not normally distributed

                Col  Probability   Statistics Rej/Acc
                  1  0.775426652  0.508683764  Accept
                  2  0.696869476  0.722314304  Accept
                  3  0.626440716  0.935402272  Accept
                  4  0.000784468  14.30100949  Reject
```

   - Shapiro-Wilks:

```
optn = [   2 ,  /* print */
         .05 ,  /* alpha */
           0 ]; /* vers */
```

```
< p4,s4 > = shapwilk(iris,optn);
print "SHAPIRO-Wilks P:",p4;
print "SHAPIRO-Wilks S:",s4;
```

```
          Shapiro-Wilks Test for Normal Distribution
       Hypothesis H0: Data are normally distributed versus
        Hypothesis H1: Data are not normally distributed

              Col  Probability   Statistics Rej/Acc
                1  0.459513152  0.101660314  Accept
                2  0.271526394  0.608203086  Accept
                3  0.054811467  1.599890243  Accept
                4  8.6586e-007  4.782450016  Reject
```

- Kolmogorov-Smirnov "cdf" version: Note that we do have to specify sample mean and standard deviation for the *cdf* transform:

```
  ari = univar(iris,"ari");
  std = univar(iris,"std");
  print "Ari=",ari," Std=",std;
```

```
Ari=
   |   Sepal_Length    Sepal_Width   Petal_Length    Petal_Width
-------------------------------------------------------------------
   |         5.0060         3.4280         1.4620        0.24600
 Std=
   |   Sepal_Length    Sepal_Width   Petal_Length    Petal_Width
-------------------------------------------------------------------
   |        0.35249        0.37906        0.17366        0.10539
```

```
 optn = [   2 ,  /* print */
          .05 ,  /* alpha */
            0 ]; /* version=cdf */
 < p3,s3 > = kstest(iris,optn,"norm",ari,std);
 print "KS-TEST P: NORM",p3;
 print "KS-TEST S: NORM",s3;
```

```
                    Kolmogorov-Smirnov Test
       Hypothesis H0: Data are normally distributed versus
        Hypothesis H1: Data are not normally distributed

              Col  Probability   Statistics Rej/Acc
                1  0.667610424  0.100742260  Accept
                2  0.779072077  0.091344209  Accept
```

```
3  0.269022215  0.138887922  Accept
4  1.6811e-005  0.335426880  Reject
```

5.8 Function `rperm`

b = rperm(a<,perm>)

Purpose: Random or specified permutation of the n rows of a matrix. This can also be done more generally by using index vectors but maybe this more specific implementation could be faster.

Input: **A** specifies an $N \times n$ matrix.

perm (optional) an N vector of integers defining the permutation applied to the N rows of matrix **A**. If `perm` is missing, it is defined by a random permutation. (Use the `srand()` for specifying a random seed.)

Output: The only output argument is the $N \times n$ matrix **B** containing the permuted rows of **A**.

Restrictions:

Relationships: cperm()

Examples:

```
print "[6] Mixed Type General Rectangular Matrix";
perm = [ 3 2 1 ];
rnam = [" row1:row3 "];
cnam = [" col1:col3 "];


amat = [ "a" 1 "c" ,
           2 "b" 2  ,
         "d" 3 "e" ];


amat = rname(amat,rnam); amat = cname(amat,cnam);
bmat = rperm(amat,perm);
```

```
                        bmat
                        ****

Mixed Type Matrix bmat

      col1 col2 col3
row3    d    3    e
row2    2    b    2
row1    a    1    c
```

The result of this index operation should be the same:

```
print "[6] RPERM=",bmat;
cmat = amat[perm,];
print "[6] RowIndex=",cmat;
```

```
                              cmat
                              ****

                    Mixed Type Matrix cmat

                          col1 col2 col3
                    row3    d    3    e
                    row2    2    b    2
                    row1    a    1    c
```

5.9 Function `rules`

< gof,rules> = rules(asso,<,optn<,cnam>>)

Purpose: The function finds rules of item purchse in with maximum *confidence* or *support*. Rules consist of two sets of items, one on the left hand side and the other on the right hand side of an arrow. Rules are selected from splitted item sets found by `assoc()` by chosing splits of large *confidence*. The number of rules can be restricted by specifying the `"minconf"` or the `"bestrul"` option. For many items the number of rules could be very large and must be reduced by specifying the `"minconf"` or `"bestrul"` options to avoid overflows. Normally, rules are sorted by descending values of *confidence*, except when the `"liftsort"` option is specified they are sorted by descending values of the *lift* value. Note, this function generates results equivalent to those of PROC RULEGEN in the SAS Enterprise Miner package. You may also get the same results by specifying the `"minconf"` or `"bestrul"` option with the `assoc()` function.

Input: **asso** This must be the output of a former run of `assoc()` with the same (or compatible) data set which must be a matrix of size $M \times (2 + nitem)$.

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See table below for content.

cnam This (optional) input argument can be a missing value or must be a vector of the names of all items which are coded with their index numbers in the input data matrix provided by `assoc()`.

Option Name	Second Column	Meaning
`"bestrul"`	int	number of rules with best *confidence*, *support*, or *lift*
`"liftsort"`		sort rules by descending lift
`"minconf"`	real	minimum *confidence* for rule generation
`"minlift"`	real	minimum *lift* for rule generation
`"minsupp"`	real	minimum *support* for rule generation
`"pini"`	int	amount printed output of input data, def=0
`"pass"`	int	print assocoiations, def=0: no output
	1 or 3	in numeric form
	2 or 3	in string form
`"prul"`	int	print rules, def=0
	1 or 3	in numeric form
	2 or 3	in string form
`"print"`	int	amount of printed output, def=0
`"suppsort"`		sort rules by descending *support*

The `"minconf"`, `"minlift"`, and `"minsupp"` specifications are combined by the logical *and* operator, that means a rule must satisfy all of them to be in the output. By default the rules are sorted by descending *confidence*, but specifying either the `"liftsort"` or the `"suppsort"` sorts them by descending *lift* or *support*.

In practice it is difficult to specify the `"minconf"` option so that the output data set contains only a small set of rules. The number of rows of the output data set is easier to specify with the `"bestrul"` option. Therefore we set `"bestrul"` as 10 times the number of items for the default.

Specifying the `"liftsort"` or the `"suppsort"` options also modifies the `"bestrul"` option in that way that such best rules are selected corresponding to the best *lift* or *support* values.

Output: **gof** a vector of scalar results

rules the $M \times (7 + nitem)$ matrix of detected rules containing in its columns:

1. the number p of all items of the rule
2. the number l of left-hand side items (the number of items on the right hand side is $r = p - l$)
3. the number c_{both} of customers who selected the complete association
4. the value of *expected confidence* of the rule

$$cexp = c_{rhs}/c_{total} * 100$$

in percent where c_{rhs} the customer count of the right subset of items and c_{total} is the total number of customers of the data

5. the value of *confidence* of the rule

$$conf = c_{both}/c_{lhs} * 100$$

in percent where c_{lhs} the customer count of the left subset of items and c_{both} is the customer count of the association with both left and right item sets united

6. the percent value of *support* of the rule

$$support = c_{both}/c_{total} * 100$$

7. the *lift* value

$$lift = c_{conf}/c_{exp}$$

8. the remaining *nitem* columns contain all the p items of the rule, first those l of the left-hand side and then those $p - l$ of the right-hand side.

Restrictions: 1. The first input argument must be a valid return object from a former `assoc()` run with the same data.

Relationships: assoc(), sequ()

Examples: 1. Example 1: SAS/EM®Data Set $N = 40, n = 3$ (see also `assoc()`):

```
ass0=[" lumber hammer nails , lumber hammer nails , lumber hammer   0   ,
        lumber hammer   0   , lumber   0    nails , lumber   0    nails ,
        lumber   0    nails , lumber   0    nails , lumber   0    nails ,
        lumber   0    nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer nails ,   0    hammer nails ,   0    hammer nails ,
          0    hammer   0   ,   0    hammer   0   ,   0    hammer   0   ,
          0    hammer   0   ,   0    hammer   0   ,   0    hammer   0   ,
          0    hammer   0   ,   0    hammer   0   ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails ,   0      0    nails ,   0      0    nails ,
          0      0    nails "];
```

```
print "Data Matrix can be binary";
ass1 = replace(ass0,"lumber",1);
ass2 = replace(ass1,"hammer",1);
ass3 = replace(ass2,"nails", 1);
print "ASS3=",ass3;
```

First, we have to run the `assoc()` function:

```
optn = [ "print"       0 ,
         "pini"        0 ,
         "pass"        0 ,
         "nitem"       3 ,
         "supp"        1 ];
< gof,asso,nset,rules > = assoc(.,ass3,optn);
```

The second output argument of `assoc()` is used as input for `rules()`:

```
optn = [ "print"       2 ,
         "pini"        2 ,
         "prul"        3 ,
         "bestrul"    10 ];
< gof,rules > = rules(asso,optn,cnam);
```

```
          Rules (10) Derived from Associations
          ************************************
```

Dense Matrix (10 by 10)

	Set_Size	LHS_Items	Count	Exp_Conf	Confidence
1	1.0000000	0	30.000000	75.000000	100.00000
2	1.0000000	0	20.000000	50.000000	100.00000
3	1.0000000	0	10.000000	25.000000	100.00000
4	2.0000000	1.0000000	8.0000000	75.000000	80.000000
5	2.0000000	1.0000000	10.000000	75.000000	50.000000
6	3.0000000	2.0000000	2.0000000	75.000000	50.000000
7	2.0000000	1.0000000	4.0000000	50.000000	40.000000
8	2.0000000	1.0000000	10.000000	50.000000	33.333333
9	2.0000000	1.0000000	8.0000000	25.000000	26.666667
10	3.0000000	2.0000000	2.0000000	50.000000	25.000000

	Support	Lift	Item_1	Item_2	Item_3
1	75.000000	1.3333333	3.0000000	.	.
2	50.000000	2.0000000	2.0000000	.	.
3	25.000000	4.0000000	1.0000000	.	.
4	20.000000	1.0666667	1.0000000	3.0000000	.
5	25.000000	0.6666667	2.0000000	3.0000000	.
6	5.0000000	0.6666667	1.0000000	2.0000000	3.0000000
7	10.000000	0.8000000	1.0000000	2.0000000	.
8	25.000000	0.6666667	3.0000000	2.0000000	.
9	20.000000	1.0666667	3.0000000	1.0000000	.
10	5.0000000	0.5000000	1.0000000	3.0000000	2.0000000

****
Rules (10) Derived from Associations
****

N	Count	Confidence	
1	30	100.000000	--> nails
2	20	100.000000	--> hammer
3	10	100.000000	--> lumber
4	8	80.0000000	lumber --> nails
5	10	50.0000000	hammer --> nails
6	2	50.0000000	lumber hammer --> nails
7	4	40.0000000	lumber --> hammer
8	10	33.3333333	nails --> hammer
9	8	26.6666667	nails --> lumber
10	2	25.0000000	lumber nails --> hammer

```
GOF=
             |         1
-------------------------
     Failure |   0.00000
   TotalTime |   0.00000
  N_Customer |    40.000
     N_Items |    3.0000
  N_Associat |    7.0000
     N_Rules |   10.0000
  SupportLev |   1.00000
  PotentSets |    7.0000
  SupLevSets |    7.0000
      unused |   0.00000
```

The following run specifies the `"minconf"` option and finds the eight rules with largest *confidence*:

```
optn = [ "print"       2 ,
         "pini"        2 ,
         "prul"        3 ,
         "minconf"   30. ];
< gof,rules > = rules(asso,optn,cnam);
print "GOF=",gof;
print "Rules=",rules;
```

Rules are sorted with decreasing *confidence* (it would also make sense to sort them by decreasing *lift*):

```
            Rules (8) Derived from Associations
            ***********************************

                   Dense Matrix (8 by 10)

    |    Set_Size  LHS_Items      Count   Exp_Conf Confidence
----------------------------------------------------------------
1 |   1.0000000          0  30.000000  75.000000  100.00000
2 |   1.0000000          0  20.000000  50.000000  100.00000
3 |   1.0000000          0  10.000000  25.000000  100.00000
4 |   2.0000000  1.0000000  8.0000000  75.000000  80.000000
5 |   2.0000000  1.0000000  10.000000  75.000000  50.000000
6 |   3.0000000  2.0000000  2.0000000  75.000000  50.000000
7 |   2.0000000  1.0000000  4.0000000  50.000000  40.000000
8 |   2.0000000  1.0000000  10.000000  50.000000  33.333333

    |     Support       Lift     Item_1     Item_2     Item_3
```

```
-------------------------------------------------------------
1 |   75.000000  1.3333333  3.0000000     .          .
2 |   50.000000  2.0000000  2.0000000     .          .
3 |   25.000000  4.0000000  1.0000000     .          .
4 |   20.000000  1.0666667  1.0000000  3.0000000     .
5 |   25.000000  0.6666667  2.0000000  3.0000000     .
6 |   5.0000000  0.6666667  1.0000000  2.0000000  3.0000000
7 |   10.000000  0.8000000  1.0000000  2.0000000     .
8 |   25.000000  0.6666667  3.0000000  2.0000000     .


                **********************************
                Rules (8) Derived from Associations
                **********************************

           N Count  Confidence
           1    30  100.000000 --> nails
           2    20  100.000000 --> hammer
           3    10  100.000000 --> lumber
           4     8  80.0000000 lumber --> nails
           5    10  50.0000000 hammer --> nails
           6     2  50.0000000 lumber hammer --> nails
           7     4  40.0000000 lumber --> hammer
           8    10  33.3333333 nails --> hammer

GOF=
              |         1
------------------------
     Failure |   0.00000
   TotalTime |   0.00000
  N_Customer |    40.000
     N_Items |    3.0000
  N_Associat |    7.0000
     N_Rules |    8.0000
      unused |   0.00000
      unused |   0.00000
      unused |   0.00000
      unused |   0.00000
```

Now we specify the `"liftsort"` option and search for the `"bestrul"`=10 rules with largest *lift* values:

```
   optn = [ "print"       2 ,
            "pini"        2 ,
            "prul"        3 ,
```

```
          "liftsort"      ,
          "bestrul"    10 ];
< gof,rules > = rules(asso,optn,cnam);
```

Now, rules are sorted by descending *lift* value:

```
           Rules (10) Derived from Associations
           ************************************

                  Dense Matrix (10 by 10)

   |    Set_Size  LHS_Items      Count   Exp_Conf Confidence
---------------------------------------------------------------
 1 |   1.0000000          0  10.000000  25.000000  100.00000
 2 |   1.0000000          0  20.000000  50.000000  100.00000
 3 |   1.0000000          0  30.000000  75.000000  100.00000
 4 |   2.0000000  1.0000000  8.0000000  25.000000  26.666667
 5 |   2.0000000  1.0000000  8.0000000  75.000000  80.000000
 6 |   2.0000000  1.0000000  4.0000000  25.000000  20.000000
 7 |   2.0000000  1.0000000  4.0000000  50.000000  40.000000
 8 |   3.0000000  2.0000000  2.0000000  25.000000  20.000000
 9 |   3.0000000  1.0000000  2.0000000  25.000000  20.000000
10 |   3.0000000  1.0000000  2.0000000  10.000000  6.6666667

   |     Support       Lift     Item_1     Item_2     Item_3
---------------------------------------------------------------
 1 |   25.000000  4.0000000  1.0000000          .          .
 2 |   50.000000  2.0000000  2.0000000          .          .
 3 |   75.000000  1.3333333  3.0000000          .          .
 4 |   20.000000  1.0666667  3.0000000  1.0000000          .
 5 |   20.000000  1.0666667  1.0000000  3.0000000          .
 6 |   10.000000  0.8000000  2.0000000  1.0000000          .
 7 |   10.000000  0.8000000  1.0000000  2.0000000          .
 8 |   5.0000000  0.8000000  2.0000000  3.0000000  1.0000000
 9 |   5.0000000  0.8000000  1.0000000  2.0000000  3.0000000
10 |   5.0000000  0.6666667  3.0000000  1.0000000  2.0000000


           ************************************
           Rules (10) Derived from Associations
           ************************************

         N Count        Lift
         1    10  4.00000000 --> lumber
         2    20  2.00000000 --> hammer
```

```
 3    30  1.33333333 --> nails
 4     8  1.06666667 nails --> lumber
 5     8  1.06666667 lumber --> nails
 6     4  0.80000000 hammer --> lumber
 7     4  0.80000000 lumber --> hammer
 8     2  0.80000000 hammer nails --> lumber
 9     2  0.80000000 lumber --> hammer nails
10     2  0.66666667 nails --> lumber hammer
```

2. Example 2: see Documentation of SAS Enterprise Miner:
   The following reads the data set which comes with the SAS/EM®package, version 9.1.3. It contains $N = 7007$ rows (sales/purchases) for $n = 20$ items.

```
  print "SAS ASSOC Data: nr=7007, nc=4";
  options NOECHO;
#include "..\\tdata\\assocs.dat"
  options ECHO;
  nr = nrow(assoc); nc = ncol(assoc);
  print "nr=",nr," nc=",nc;


   ass0 = assoc[,2:4];
   ass1 = replace(ass0,"apples",1);
   ass0 = replace(ass1,"artichok",2);
   ass1 = replace(ass0,"avocado",3);
   ass0 = replace(ass1,"baguette",4);
   ass1 = replace(ass0,"bordeaux",5);
   ass0 = replace(ass1,"bourbon",6);
   ass1 = replace(ass0,"chicken",7);
   ass0 = replace(ass1,"coke",8);
   ass1 = replace(ass0,"corned_b",9);
   ass0 = replace(ass1,"cracker",10);
   ass1 = replace(ass0,"ham",11);
   ass0 = replace(ass1,"heineken",12);
   ass1 = replace(ass0,"hering",13);
   ass0 = replace(ass1,"ice_crea",14);
   ass1 = replace(ass0,"olives",15);
   ass0 = replace(ass1,"peppers",16);
   ass1 = replace(ass0,"sardines",17);
   ass0 = replace(ass1,"soda",18);
   ass1 = replace(ass0,"steak",19);
   ass0 = replace(ass1,"turkey",20);


   cust = ass0[,1]; vist = ass0[,2];
```

```
nv = 20;
ind1 = [ 1 : nr ]; ind2 = ass0[,3]; ind = ind1‘ -> ind2;
/* print "Ind=", ind[1:10,]; */
ass1 = cons(nr,nv,0.);
ass1[ind] = 1;


cnam = [" apples   artichok  avocado  baguette  bordeaux
          bourbon  chicken   coke     corned_b  cracker
          ham      heineken  hering   ice_crea  olives
          peppers  sardines  soda     steak     turkey "];
ass1 = cname(ass1,cnam);
print "Sparse Binary AssoData for Input=",ass1[1:10,];
attrib(ass1); /* free ass0; */
```

The input data are binary and very sparse:

```
---------------------------------------
         Table of Attributes
---------------------------------------
Object Name      name             ass1
Object Type      otyp       matrix_gen
Data Type        dtyp              int
Storage Type     styp        spar_full
Row Names        rnam                0
Column Names     cnam               20
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow             7007
Number Columns   ncol               20
Lower Bandwidth  lbw              7002
Upper Bandwidth  ubw                15
Size in Bytes    size           112424
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer             7007
Smallest Value   vmin                0
Largest Value    vmax                1
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
---------------------------------------
```

We select up to 5 items with a minumum *support* of 20 as shown in the documentation of the SAS PROC ASSOC:

```
optn = [ "print"     0 ,
         "pini"      0 ,
         "pass"      0 ,
         "nitem"     5 ,
         "supp"     20 ];
< gof,asso,nset > = assoc(cust,ass1,optn);
```

For the results see the document of the `assoc()` function. Use the output `asso` as input argument for the `rules()` function and require *confidence* values of at least 75

```
optn = [ "print"       2 ,
         "pini"        2 ,
         "pass"        3 ,
         "prul"        3 ,
         "liftsort"      ,
         "minconf"   75. ];
< gof,rules > = rules(asso,optn,cnam);
print "GOF=",gof;
print "Rules=",rules;
```

We obtain 939 rules sorted with descending *lift* value. The results agree with those shown in the document for the SAS PROC RULEGEN:

```
          Rules (939) Derived from Associations
          *************************************

                Dense Matrix (939 by 12)

    |    Set_Size  LHS_Items      Count   Exp_Conf Confidence
----------------------------------------------------------
  1 |   1.0000000          0  74.000000  7.3926074  100.00000
  2 |   5.0000000  3.0000000  90.000000  12.587413  94.736842
  3 |   5.0000000  2.0000000  94.000000  10.789211  78.991597
  4 |   5.0000000  3.0000000  94.000000  11.888112  87.037037
  5 |   5.0000000  3.0000000  90.000000  12.687313  92.783505
  6 |   5.0000000  3.0000000  90.000000  13.186813  93.750000
  7 |   5.0000000  3.0000000  90.000000  13.886114  97.826087
  8 |   5.0000000  3.0000000  90.000000  12.587413  88.235294
  9 |   5.0000000  3.0000000  90.000000  13.886114  96.774194
```

10	5.0000000	3.0000000	116.00000	13.986014	96.666667
935	4.0000000	3.0000000	31.000000	59.940060	75.609756
936	2.0000000	1.0000000	366.00000	59.940060	75.000000
937	4.0000000	3.0000000	24.000000	59.940060	75.000000
938	4.0000000	3.0000000	24.000000	59.940060	75.000000
939	3.0000000	2.0000000	45.000000	59.940060	75.000000

	Support	Lift	Item_1	Item_2	Item_3
1	7.3926074	13.527027	5.0000000	.	.
2	8.9910090	7.5263158	1.0000000	4.0000000	17.000000
3	9.3906094	7.3213508	8.0000000	20.000000	6.0000000
4	9.3906094	7.3213508	6.0000000	14.000000	15.000000
5	8.9910090	7.3130936	1.0000000	4.0000000	16.000000
6	8.9910090	7.1093750	1.0000000	3.0000000	16.000000
7	8.9910090	7.0448858	1.0000000	3.0000000	17.000000
8	8.9910090	7.0098039	3.0000000	4.0000000	16.000000
9	8.9910090	6.9691344	4.0000000	16.000000	17.000000
10	11.588412	6.9116667	8.0000000	12.000000	17.000000
935	3.0969031	1.2614228	2.0000000	6.0000000	10.000000
936	36.563437	1.2512500	10.000000	12.000000	.
937	2.3976024	1.2512500	2.0000000	3.0000000	17.000000
938	2.3976024	1.2512500	2.0000000	3.0000000	16.000000
939	4.4955045	1.2512500	2.0000000	14.000000	12.000000

	Item_4	Item_5
1	.	.
2	3.0000000	16.000000
3	14.000000	15.000000
4	8.0000000	20.000000
5	3.0000000	17.000000
6	4.0000000	17.000000
7	4.0000000	16.000000
8	1.0000000	17.000000
9	1.0000000	3.0000000
10	7.0000000	14.000000
935	12.000000	.
936	.	.
937	12.000000	.
938	12.000000	.
939	.	.

```
                  ************************************
                  Rules (939) Derived from Associations
                  ************************************

   N Count        Lift
   1    74  13.5270270 --> bordeaux
   2    90  7.52631579 apples baguette sardines --> avocado peppers
   3    94  7.32135076 coke turkey --> bourbon ice_crea olives
   4    94  7.32135076 bourbon ice_crea olives --> coke turkey
   5    90  7.31309360 apples baguette peppers --> avocado sardines
   6    90  7.10937500 apples avocado peppers --> baguette sardines
   7    90  7.04488583 apples avocado sardines --> baguette peppers
   8    90  7.00980392 avocado baguette peppers --> apples sardines
   9    90  6.96913437 baguette peppers sardines --> apples avocado
  10   116  6.91166667 coke heineken sardines --> chicken ice_crea
  ..............................................................
 935    31  1.26142276 artichok bourbon cracker --> heineken
 936   366  1.25125000 cracker --> heineken
 937    24  1.25125000 artichok avocado sardines --> heineken
 938    24  1.25125000 artichok avocado peppers --> heineken
 939    45  1.25125000 artichok ice_crea --> heineken


GOF=
             |         1
-------------------------
     Failure |   0.00000
   TotalTime |   0.00000
  N_Customer |    1001.0
     N_Items |    20.000
  N_Associat |    1206.0
     N_Rules |    939.00
      unused |   0.00000
      unused |   0.00000
      unused |   0.00000
      unused |   0.00000
```

5.10 Function `sequ`

< gof,nset,oseq1,oseq2,...> = sequ(cust,visit,data,asso<,optn<,supp>>)

Purpose: This function uncovers important sequences in the timely purchase behavior of customers. Similar to the `assoc()` function the input data must contain,

- a column `cust` with the int or real customer number
- the (usually very sparse) `data` matrix with `nitm` columns (with binary information) indicating which items where purchased
- and in addition to `assoc()` it also needs a column with real values indicating the time of purchase.

Therefore, the `sequ` function does not only answer the question: if a customer buys a computer which other items, like a printer, he would probably purchase in addition, it also answers if he would purchase those other items within of a specific timely interval. Note, this function generates results equivalent to those of PROC SEQUENCE in the SAS Enterprise Miner package.

Input: **cust** This must be an N vector of integers defining the customer IDs used in N purchases. There could be multiple purchases by the same customer ID. It is not necessary that the N observations of the first two input arguments are sorted w.r.t. ascending customer IDs, the `assoc()` function would do that at the input.

visit This must be an N vector of reals defining the visits (times) used in N purchases. There could be multiple visits (times) by the same customer ID. The visit times are important only w.r.t. the same customer number and if they are not sorted before input into `sequ()` they will be sorted within groups of the the same customer. If the visit times are the same for two purchses of the same customer the rows of the binary data are combined with the union operator. It is not necessary that the N observations of the first two input arguments are sorted w.r.t. ascending customer IDs, the `assoc()` function would do that at the input.

data This must be an $N \times n$ matrix of N purchases of n items, where `data[i,j]=1` stands for the purchase of item j by customer `cust[i]` and `data[i,j]=0` means that item j was not bought at the purchase i. This matrix is usually very sparse and maybe binary. If the data values are not binary, the `assoc()` function will transform them into binary data at the input. The binary purchase rows with the same customer ID are then combined with the union operator.

asso This must be the output of a former run of `assoc()` with the same (or compatible) data set which must be a matrix of size $M \times (2 + nitem)$.

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See table below for content.

supp If this optional argument is not missing, it must be an n vector of integers defining the minimum *support* of each item at the first stage of associations. If the vector contain missing values the will be relaced by the value specified with the `"supp"` option.

supp

Options Matrix Argument: The option argument is specified in form of a two column matrix:

Option Name	Second Column	Meaning
`"nitem"`	int	maximum number of item sets of a sequence
`"pcts"`	real	percentage of minimum *support* in each stage
`"pout"`	int	print sequence matrix, def=0: no output
	1 or 3	in numeric form
	2 or 3	in string form
`"print"`	int	the amount of printed output, def=0
`"same"`	real	specifies the lower time limit between the occurence of two purchases which may be associated with each other
`"supp"`	int	minimum *support* in each stage
`"suppsort"`		sort output sequences by descending *support*
`"window"`	real	specifies the maximum time difference between the occurence of two purchases which is to be treated as the same visit

The value for `"nitem"` is here defined by the `asso` input argument. The first observation of the `asso` input argument also states the number of customers which is compared to the number of customers defined by the `data` input argument. For the first stage of associations the nonmissing values of the itemwise support vector specification (optional input argument 4) overrides the more general options specifications.

Output: gof a vector of scalar results

nset an $item \times 3$ matrix with the numbers of all sequences and those meeting the support level. The last column refers to the column numbers of the corresponding `oseq.` output argument which is the maximum number of items used in the rules. The first row gives the number of rows of the input `asso` matrix.

oseq1,oseq2,... for $item = 2, 3, \ldots$ the matrices containing the sequences with the following columns:

1. customer count of the sequence
2. value of *support*

3. value of *confidence*
4. the number k of items in the left rule
5. the k items in the left rule
6. .....
7. the number k of items in the right rule
8. the k items in the right rule

By default the rows of the matrix are sorted by descending *confidence* (as with the `rules()` function) which can be changed specifying the `"suppsort"` option which will order the rows by descending *support* value.

Restrictions: 1. Strings, missing values, and complex data are not permitted as input of the first two arguments.

2. The first four input arguments must have compatible sizes.

Relationships: assoc(), rules()

Examples: 1. Example 1: see Documentation of SAS Enterprise Miner:

The following reads the data set which comes with the SAS/EM®package, version 9.1.3. It contains $N = 7007$ rows (sales/purchases) for $n = 20$ items.

```
  print "SAS ASSOC Data: nr=7007, nc=4";
  options NOECHO;
#include "..\\tdata\\assocs.dat"
  options ECHO;
  nr = nrow(assoc); nc = ncol(assoc);
  print "nr=",nr," nc=",nc;


   ass0 = assoc[,2:4];
   ass1 = replace(ass0,"apples",1);
   ass0 = replace(ass1,"artichok",2);
   ass1 = replace(ass0,"avocado",3);
   ass0 = replace(ass1,"baguette",4);
   ass1 = replace(ass0,"bordeaux",5);
   ass0 = replace(ass1,"bourbon",6);
   ass1 = replace(ass0,"chicken",7);
   ass0 = replace(ass1,"coke",8);
   ass1 = replace(ass0,"corned_b",9);
   ass0 = replace(ass1,"cracker",10);
   ass1 = replace(ass0,"ham",11);
   ass0 = replace(ass1,"heineken",12);
   ass1 = replace(ass0,"hering",13);
   ass0 = replace(ass1,"ice_crea",14);
   ass1 = replace(ass0,"olives",15);
```

```
ass0 = replace(ass1,"peppers",16);
ass1 = replace(ass0,"sardines",17);
ass0 = replace(ass1,"soda",18);
ass1 = replace(ass0,"steak",19);
ass0 = replace(ass1,"turkey",20);


cust = ass0[,1]; vist = ass0[,2];
nv = 20;
ind1 = [ 1 : nr ]; ind2 = ass0[,3]; ind = ind1‘ -> ind2;
/* print "Ind=", ind[1:10,]; */
ass1 = cons(nr,nv,0.);
ass1[ind] = 1;


cnam = [" apples   artichok  avocado  baguette  bordeaux
          bourbon  chicken   coke     corned_b  cracker
          ham      heineken  hering   ice_crea  olives
          peppers  sardines  soda     steak     turkey "];
ass1 = cname(ass1,cnam);
print "Sparse Binary AssoData for Input=",ass1[1:10,];
attrib(ass1); /* free ass0; */
```

The input data are binary and very sparse:

```
--------------------------------------
         Table of Attributes
--------------------------------------
Object Name      name             ass1
Object Type      otyp       matrix_gen
Data Type        dtyp              int
Storage Type     styp        spar_full
Row Names        rnam                0
Column Names     cnam               20
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow             7007
Number Columns   ncol               20
Lower Bandwidth  lbw              7002
Upper Bandwidth  ubw                15
Size in Bytes    size           112424
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer             7007
Smallest Value   vmin                0
```

```
Largest Value    vmax                1
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
-------------------------------------
```

We select up to 5 items with a minumum *support* of 20 as shown in the documentation of the SAS PROC ASSOC:

```
optn = [ "print"     1 ,
         "pini"      1 ,
         "pass"      3 ,
         "nitem"     5 ,
         "supp"     20 ];
< gof,asso,nset > = assoc(cust,ass1,optn);
```

For the results see the document of the `assoc()` function. Use the output `asso` as input argument for the `rules()` function and require *confidence* values of at least 75

```
print "SAS document for PROC SEQUENCE: Example 1";
optn = [ "print"     2 ,
         "pout"      3 ,
         "suppsor"      ,
         "supp"     20 ,
         "nitem"     2 ];
< gof,nset,sequs > = sequ(cust,vist,ass1,asso,optn);
print "GOF=",gof;
print "Nset=",nset;
```

The results agree with those documented for the PROC SEQUENCE in the SAS Enterprice Miner package:

```
              Sequences (291) With 2 Item Sets
              ********************************

                  Dense Matrix (291 by 7)

    |       Count    Support Confidence    NITEM_1    RUL_1_1
----------------------------------------------------------------
  1 |   337.00000  33.666334  69.057377  1.0000000  10.000000
  2 |   235.00000  23.476523  48.353909  1.0000000  13.000000
```

```
  3 |   233.00000  23.276723  49.260042  1.0000000  15.000000
  4 |   229.00000  22.877123  47.119342  1.0000000  13.000000
  5 |   226.00000  22.577423  46.502058  1.0000000  13.000000
  6 |   225.00000  22.477522  57.397959  1.0000000  4.0000000
  7 |   220.00000  21.978022  69.182390  1.0000000  18.000000
  8 |   220.00000  21.978022  56.122449  1.0000000  4.0000000
  9 |   220.00000  21.978022  46.511628  1.0000000  15.000000
 10 |   218.00000  21.778222  68.553459  1.0000000  18.000000
 ..............................................................
287 |   20.000000  1.9980020  5.1150895  1.0000000  9.0000000
288 |   20.000000  1.9980020  6.7567568  1.0000000  16.000000
289 |   20.000000  1.9980020  7.0671378  1.0000000  20.000000
290 |   20.000000  1.9980020  8.8105727  1.0000000  19.000000
291 |   20.000000  1.9980020  8.8105727  1.0000000  19.000000

    |     NITEM_2    RUL_2_1
 ---------------------------
  1 |   1.0000000  12.000000
  2 |   1.0000000  12.000000
  3 |   1.0000000  6.0000000
  4 |   1.0000000  9.0000000
  5 |   1.0000000  15.000000
  6 |   1.0000000  12.000000
  7 |   1.0000000  10.000000
  8 |   1.0000000  13.000000
  9 |   1.0000000  20.000000
 10 |   1.0000000  12.000000
 ........................
287 |   1.0000000  5.0000000
288 |   1.0000000  20.000000
289 |   1.0000000  3.0000000
290 |   1.0000000  7.0000000
291 |   1.0000000  11.000000

              ******************************
              Sequences (291) With 2 Item Sets
              ******************************

   N Count     Support  Confidence
   1   337  33.6663337  69.0573770 cracker --> heineken
   2   235  23.4765235  48.3539095 herring --> heineken
   3   233  23.2767233  49.2600423 olives --> bourbon
   4   229  22.8771229  47.1193416 herring --> corned_b
   5   226  22.5774226  46.5020576 herring --> olives
   6   225  22.4775225  57.3979592 baguette --> heineken
```

```
  7   220  21.9780220  69.1823899 soda --> cracker
  8   220  21.9780220  56.1224490 baguette --> herring
  9   220  21.9780220  46.5116279 olives --> turkey
 10   218  21.7782218  68.5534591 soda --> heineken
 ..................................................
287    20  1.99800200  5.11508951 corned_b --> bordeaux
288    20  1.99800200  6.75675676 peppers --> turkey
289    20  1.99800200  7.06713781 turkey --> avocado
290    20  1.99800200  8.81057269 steak --> chicken
291    20  1.99800200  8.81057269 steak --> ham


                    Number of Customers: 1001
                Maximum number visits per customer: 7
                Maximum number items per customer: 7
                 Number of Input Associations: 1206
                           Support Level: 20
                      Total Number of Sequences: 398
                Sequences Meeting Support Level: 291
                      Total Processing Time: 1


GOF=
             |         1
-------------------------
     Failure |   0.00000
   TotalTime |   1.00000
  N_Customer |    1001.0
  N_Associat |    1206.0
  SupportLev |    20.000
  MaxVisitpC |    7.0000
  MaxItemspC |    7.0000
  NTotalSequ |    398.00
  NSuppLSequ |    291.00
      unused |   0.00000


Nset=
          |   Total_N_Sequ   SuppLev_Sequ      Max_Items
----------------------------------------------------------
  N_Assoc |           1206           1206              5
   Item_2 |            398            291              1
```

2. Example 2: see Documentation of SAS Enterprise Miner:
   The following run of `sequ()` with the same data is also from the SAS document of PROC SEQUENCE and specifies four item sets and a

"same" value of 2. The "same" option treats successive visits with a time difference $B - A$ smaller or equal to the specified value as the same visit. When the time difference $B - A$ of successive visits is larger than the specified value of "window" a sequence will not be considered. That means only visits with $same < B - A \leq window$ are being considered for sequences.

```
print "SAS document for PROC SEQUENCE: Example 2";
optn = [ "print"     2 ,
         "pout"      3 ,
         "suppsor"     ,
         "supp"     20 ,
         "same"      2 ,
         "nitem"     4 ];
< gof,nset,seq2,seq3,seq4 > = sequ(cust,vist,ass1,asso,optn);
print "GOF=",gof;
print "NSET=",nset;
```

Sequences (463) With 2 Item Sets

Dense Matrix (463 by 15)

	Count	Support	Confidence	NITEM_1	RUL_1_1
1	235.00000	23.476523	48.353909	1.0000000	13.000000
2	225.00000	22.477522	57.397959	1.0000000	4.0000000
3	220.00000	21.978022	69.182390	1.0000000	18.000000
4	218.00000	21.778222	68.553459	1.0000000	18.000000
5	218.00000	21.778222	68.553459	1.0000000	18.000000
6	215.00000	21.478521	45.454545	1.0000000	15.000000
7	213.00000	21.278721	52.853598	1.0000000	6.0000000
8	209.00000	20.879121	100.00000	3.0000000	4.0000000
9	201.00000	20.079920	55.371901	1.0000000	3.0000000
10	150.00000	14.985015	30.864198	1.0000000	13.000000

	RUL_1_2	RUL_1_3	RUL_1_4	RUL_1_5	NITEM_2
1	.	.	.	.	1.0000000
2	.	.	.	.	1.0000000
3	.	.	.	.	1.0000000
4	.	.	.	.	2.0000000
5	.	.	.	.	1.0000000
6	.	.	.	.	1.0000000

```
 7 |      .          .          .          .       1.0000000
 8 |   7.0000000  12.000000     .          .       1.0000000
 9 |      .          .          .          .       1.0000000
10 |      .          .          .          .       1.0000000
...........................................................
```

```
   |     RUL_2_1    RUL_2_2    RUL_2_3    RUL_2_4    RUL_2_5
-----------------------------------------------------------
 1 |   12.000000     .          .          .          .
 2 |   12.000000     .          .          .          .
 3 |   10.000000     .          .          .          .
 4 |   10.000000  12.000000     .          .          .
 5 |   12.000000     .          .          .          .
 6 |   20.000000     .          .          .          .
 7 |   10.000000     .          .          .          .
 8 |   12.000000     .          .          .          .
 9 |   12.000000     .          .          .          .
10 |   10.000000     .          .          .          .
...........................................................
```

```
                ********************************
                Sequences (463) With 2 Item Sets
                ********************************

 N Count     Support
 1   235  23.4765235 herring --> heineken
 2   225  22.4775225 baguette --> heineken
 3   220  21.9780220 soda --> cracker
 4   218  21.7782218 soda --> cracker heineken
 5   218  21.7782218 soda --> heineken
 6   215  21.4785215 olives --> turkey
 7   213  21.2787213 bourbon --> cracker
 8   209  20.8791209 baguette chicken heineken --> heineken
 9   201  20.0799201 avocado --> heineken
10   150  14.9850150 herring --> cracker
```

```
              Sequences (12) With 3 Item Sets
              ********************************

                  Dense Matrix (12 by 12)

   |       Count    Support Confidence    NITEM_1    RUL_1_1
-------------------------------------------------------------
 1 |   23.000000  2.2977023  11.442786  1.0000000  3.0000000
```

2	22.000000	2.1978022	10.000000	1.0000000	18.000000
3	22.000000	2.1978022	10.091743	1.0000000	18.000000
4	22.000000	2.1978022	10.328638	1.0000000	6.0000000
5	22.000000	2.1978022	10.091743	1.0000000	18.000000
6	21.000000	2.0979021	9.8591549	1.0000000	6.0000000
7	21.000000	2.0979021	9.3333333	1.0000000	4.0000000
8	20.000000	1.9980020	9.3023256	1.0000000	15.000000
9	20.000000	1.9980020	9.3023256	1.0000000	15.000000
10	20.000000	1.9980020	9.1743119	1.0000000	18.000000
11	20.000000	1.9980020	9.1743119	1.0000000	18.000000
12	20.000000	1.9980020	9.0909091	1.0000000	18.000000

	RUL_1_2	NITEM_2	RUL_2_1	RUL_2_2	NITEM_3
1	.	1.0000000	12.000000	.	1.0000000
2	.	1.0000000	10.000000	.	1.0000000
3	.	1.0000000	12.000000	.	1.0000000
4	.	1.0000000	10.000000	.	1.0000000
5	.	2.0000000	10.000000	12.000000	1.0000000
6	.	1.0000000	10.000000	.	1.0000000
7	.	1.0000000	12.000000	.	1.0000000
8	.	1.0000000	20.000000	.	1.0000000
9	.	1.0000000	20.000000	.	1.0000000
10	.	1.0000000	12.000000	.	1.0000000
11	.	2.0000000	10.000000	12.000000	1.0000000
12	.	1.0000000	10.000000	.	1.0000000

	RUL_3_1	RUL_3_2
1	9.0000000	.
2	7.0000000	.
3	7.0000000	.
4	13.000000	.
5	7.0000000	.
6	1.0000000	.
7	9.0000000	.
8	4.0000000	.
9	1.0000000	.
10	3.0000000	.
11	3.0000000	.
12	3.0000000	.

****
Sequences (12) With 3 Item Sets

```
                    ******************************

      N Count     Support
      1    23  2.29770230 avocado --> heineken --> corned_b
      2    22  2.19780220 soda --> cracker --> chicken
      3    22  2.19780220 soda --> heineken --> chicken
      4    22  2.19780220 bourbon --> cracker --> herring
      5    22  2.19780220 soda --> cracker heineken --> chicken
      6    21  2.09790210 bourbon --> cracker --> apples
      7    21  2.09790210 baguette --> heineken --> corned_b
      8    20  1.99800200 olives --> turkey --> baguette
      9    20  1.99800200 olives --> turkey --> apples
     10    20  1.99800200 soda --> heineken --> avocado
     11    20  1.99800200 soda --> cracker heineken --> avocado
     12    20  1.99800200 soda --> cracker --> avocado


                    Number of Customers: 1001
                  Maximum number visits per customer: 3
                  Maximum number items per customer: 15
                   Number of Input Associations: 1206
                            Support Level: 20
                      Total Number of Sequences: 0
                   Sequences Meeting Support Level: 0
                        Total Processing Time: 2

GOF=
              |         1
------------------------
     Failure |   0.00000
   TotalTime |    2.0000
  N_Customer |    1001.0
  N_Associat |    1206.0
  SupportLev |    20.000
  MaxVisitpC |    3.0000
  MaxItemspC |    15.000
  NTotalSequ |   0.00000
  NSuppLSequ |   0.00000
      unused |   0.00000


NSET=
           |   Total_N_Sequ   SuppLev_Sequ      Max_Items
-----------------------------------------------------------
  N_Assoc |           1206           1206              5
   Item_2 |           5657            463              5
```

Item_3	5053	12	2
Item_4	0	0	0

5.11 Function `voronoi`

< gof,vert,offs,cent,neigh> = voronoi(x<,optn<,thresh<,bounds> . >)

Purpose: The function computes the voronoi diagrams for a small dimensional data set x. The `qhull` package is being used (Barber, Dobkin, & Huhdanpaa, 1996) as it is used in Matlab, Octave, and the R package `geometry`. (Please, see the copyright notice.)

Input: **x** Should be a $npnt \times ndim$ matrix of real data, containing the coordinates of $npnt > ndim$ point in $ndim \geq 2$ dimensions, and must not contain any string, complex, or missing data.

optn This argument must be specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See the large table given for the convhull() function for content.

tresh must be a $ndim \times 2$ real matrix of thresholds, default is $[-MACBIG, MACBIG]$ for missing values

bounds must be a $ndim \times 2$ real matrix of bounds, default is $[-MACBIG, MACBIG]$ for missing values

Options Matrix Argument: The option argument is specified in form of a two column matrix:

Output: **gof** column vector with some scalar results.

vert $nfac \times ndim$ matrix of the vertex points of $nfac$ facets in form of row indices of the input points x.

offs $nfac \times 3$ matrix with facet offset, inner, and facet area

cent $nfac \times ndim$ matrix of facet centers

neigh an $nfac \times K$ matrix with indices of facet neighbors

Restrictions:
1. Currently no missing values, complex data, or strings are permitted in input argument x.
2. For compatibility of options see the `qhull` software.

Relationships: convhull(), delaunay()

Examples: 1. Example 1: nr=10, nc=3:

```
ex2 = [ -0.4243823156725032   0.04819078870788251  0.2599563386958589  ,
         0.05173940739671138  0.1290291911430998  -0.4802858540027955  ,
         0.3968302858716195   0.1627966372328745   0.2569493707342297  ,
         0.254926174024976    0.3755187785602292  -0.2097577000866027  ,
         0.09739528838103902  0.3150142760660069  -0.3758725364745126  ,
```

```
        -0.3258558001657499   0.3443313655641816  -0.1589147827832413  ,
        -0.08840351591503222  0.3534152565992987  -0.3424652898860154  ,
        -0.07566072217015557 -0.2858472097010178   0.4031957686122621  ,
        -0.4661824453740068  -0.1807514924031959  -0.001680957800978008,
         0.05219712128711454  0.3653971507699565  -0.3372838311252646  ];


 print "VORONOI: 2-column matrix options: NO Qz option";
 print "rbox 10 s | qvoron_11 f i p s Fv Fn FN Fx";
 optn = [ "pfac"       , /* f:  5: print facets */
          "pinci"      , /* i: 14: vertices incident to each factor */
          "pnorm"      , /* n: 11: print normals with offsets */
          "ppoi"       , /* p: 21: print point coordinates */
          "psum"       , /* s: 24: print summary */
          "pneig"      , /* Fn: print neighbors */
          "pvneig"     , /* FN: print neighbor vertex */
          "pvave"      , /* Fv: print vertex average */
          "pextr"      , /* Fx: print extremes */
          "pstat"      ,
          "print"    4 ];
 < gof,vert,offs,cent,neigh > = voronoi(ex2,optn);


  VORONOI() invoked by options:   Pfacets  Pincidence  Pnormals
Ppoints  Psummary  Pneighbors  PNeighbors-vertex  PVertex-average
  PExtremes  Pstatistics  QbBound-unit-box  Qcoplanar-keep  Farea


 Analyzed Data Points: Number Points=10 Dimension=4
 -------------------------------------------------

   1|     -0.4244     0.04819        0.26        0.25
   2|     0.05174       0.129     -0.4803        0.25
   3|      0.3968      0.1628      0.2569        0.25
   4|      0.2549      0.3755     -0.2098        0.25
   5|      0.0974       0.315     -0.3759        0.25
   6|     -0.3259      0.3443     -0.1589        0.25
   7|     -0.0884      0.3534     -0.3425        0.25
   8|    -0.07566     -0.2858      0.4032        0.25
   9|     -0.4662     -0.1808   -0.001681        0.25
  10|      0.0522      0.3654     -0.3373        0.25


     Voronoi diagram by the convex hull of 10 points in 4-d:
     *******************************************************

  Options:   Pfacets  Pincidence  Pnormals  Ppoints  Psummary
```

```
   Pneighbors  PNeighbors-vertex  PVertex-average  PExtremes
                          Pstatistics
 Number of Voronoi regions . . . . . .          10
 Number of Voronoi vertices. . . . . .          12
 Number of points processed. . . . . .          10
 Number of hyperplanes created . . . . .        44
 Number of facets in hull. . . . . .            25
 Number of distance tests for qhull. . . . .    63
 CPU seconds to compute hull (after input) . . . . .  0
 Total facet area. . . . . .                   0.1
```

The "pvave" output:

```
                          Vertex Average
                          --------------

             -0.05274      0.1627    -0.09862         0.5
```

The "pfac" output:

```
                  Vertices (10) and Facets (25)
                  -----------------------------

- p5 (v8): -0.33  0.34 -0.16  0.25
  neighbors: f19 f20 f21 f24 f25 f27 f28 f34 f35 f37
             f39 f40 f41 f43
- p0 (v6): -0.42  0.048  0.26   0.5
  neighbors: f7 f14 f19 f21 f25 f27
- p7 (v4): -0.076 -0.29   0.4     0
  neighbors: f3 f7 f19 f24 f25 f28 f30 f35
- p2 (v1):   0.4  0.16  0.26   0.5
  neighbors: f3 f7 f12 f13 f14 f19 f20 f21 f29 f30
             f34 f35
- p1 (v3):  0.052  0.13 -0.48   0.5
  neighbors: f3 f12 f13 f24 f28 f29 f30 f31 f36 f37
             f38 f39
- p8 (v0): -0.47 -0.18 -0.0017     1
  neighbors: f3 f7 f12 f14 f24 f25 f27 f38 f39 f40
- p4 (v9):  0.097  0.32 -0.38  0.25
  neighbors: f28 f29 f30 f31 f34 f35 f36 f37 f42 f43
- p3 (v2):  0.25  0.38 -0.21   0.5
  neighbors: f13 f20 f29 f31 f34 f41 f42 f43
- p6 (v10): -0.088  0.35 -0.34   0.5
  neighbors: f36 f37 f38 f39 f40 f41 f42 f43
- p9 (v7):  0.052  0.37 -0.34     1
```

```
  neighbors: f12 f13 f14 f20 f21 f27 f31 f36 f38 f40
             f41 f42


- f19
    - flags: top simplicial
    - area: 0.016
    - normal:     -0.09198      0.6762      0.6528     -0.3289
    - offset: -0.07685344
    - center:    1.11e-016  1.943e-016  2.776e-016
    - vertices: p5 (v8) p0 (v6) p7 (v4) p2 (v1)
    - neighboring facets:  f7 f35 f21 f25

- f24
    - flags: bottom simplicial
    - area: 0.028
    - normal:      -0.5268     -0.5448     -0.5395     -0.3669
    - offset: 0.02191342
    - center:  -1.665e-016 -1.665e-016 -1.756e-016
    - vertices: p5 (v8) p7 (v4) p1 (v3) p8 (v0)
    - neighboring facets:  f3 f39 f25 f28

- f25
    - flags: bottom simplicial
    - area: 0.013
    - normal:       -0.807     -0.3889     -0.2425     -0.3724
    - offset: -0.07445756
    - center:   -2.22e-016  -1.11e-016 -8.218e-017
    - vertices: p5 (v8) p0 (v6) p7 (v4) p8 (v0)
    - neighboring facets:  f7 f24 f27 f19


- f28
    - flags: bottom simplicial
    - area: 0.015
    - normal:        -0.33     -0.4124      -0.588     -0.6126
    - offset: 0.09421304
    - center:            0  -1.11e-016  -1.11e-016
    - vertices: p4 (v9) p5 (v8) p7 (v4) p1 (v3)
    - neighboring facets:  f24 f30 f37 f35

- f29
    - flags: top simplicial
    - area: 0.0018
    - normal:       0.8529     -0.2789     -0.3865     -0.2131
    - offset: -0.08717703
    - center:   3.886e-016  -2.22e-016  -2.22e-016
```

```
    - vertices: p4 (v9) p1 (v3) p3 (v2) p2 (v1)
    - neighboring facets:  f13 f34 f30 f31

- f30
    - flags: top simplicial
    - area: 0.012
    - normal:        0.734     -0.4798     -0.3216     -0.3572
    - offset: 0.04806399
    - center:   3.331e-016 -1.665e-016           0
    - vertices: p4 (v9) p7 (v4) p1 (v3) p2 (v1)
    - neighboring facets:  f3 f29 f35 f28


- f34
    - flags: bottom simplicial
    - area: 0.0063
    - normal:       0.2045      0.7804      0.2935     -0.5128
    - offset: -0.02723847
    - center:  -5.551e-017 -3.331e-016 -5.551e-017
    - vertices: p4 (v9) p5 (v8) p3 (v2) p2 (v1)
    - neighboring facets:  f20 f29 f35 f43

- f35
    - flags: top simplicial
    - area: 0.028
    - normal:       0.2079      0.6345      0.3198     -0.6722
    - offset: 0.06814782
    - center:            0  1.665e-016  8.327e-017
    - vertices: p4 (v9) p5 (v8) p7 (v4) p2 (v1)
    - neighboring facets:  f19 f30 f34 f28

- f37
    - flags: top simplicial
    - area: 0.0011
    - normal:      -0.4132      0.2585     -0.8411     -0.2344
    - offset: -0.2987272
    - center:  -2.776e-016  1.665e-016 -5.551e-016
    - vertices: p6 (v10) p4 (v9) p5 (v8) p1 (v3)
    - neighboring facets:  f28 f39 f36 f43


- f41
    - flags: bottom simplicial
    - area: 6.4e-005
    - normal:     -0.05246      0.9981    -0.03052    -0.00885
    - offset: -0.3634156
    - center:  -6.106e-016  1.193e-014 -2.776e-016
```

```
    - vertices: p6 (v10) p5 (v8) p9 (v7) p3 (v2)
    - neighboring facets:  f20 f42 f43 f40

- f42
    - flags: top simplicial
    - area: 0.00016
    - normal:      0.07573      0.9324     -0.3512       -0.04
    - offset: -0.4230989
    - center:   5.551e-017  1.776e-015 -6.106e-016
    - vertices: p6 (v10) p4 (v9) p9 (v7) p3 (v2)
    - neighboring facets:  f31 f41 f43 f36

- f43
    - flags: bottom simplicial
    - area: 0.00076
    - normal:    -0.006787      0.9799     -0.1457     -0.1361
    - offset:  -0.328746
    - center:  -1.665e-016  1.055e-015 -3.608e-016
    - vertices: p6 (v10) p4 (v9) p5 (v8) p3 (v2)
    - neighboring facets:  f34 f41 f42 f37
```

The "pneig" output:

```
                    Neighbors of 12 Facets
                    ----------------------

                     1 4 :  -7   7 -21   2
                     2 4 :  -3 -39   2   3
                     3 4 :  -7   1 -27   0
                     4 4 :   1   5   8   7
                     5 4 : -13   6   5 -31
                     6 4 :  -3   4   7   3
                     7 4 : -20   4   7  11
                     8 4 :   0   5   6   3
                     9 4 :   3 -39 -36  11
                    10 4 : -20  10  11 -40
                    11 4 : -31   9  11 -36
                    12 4 :   6   9  10   8
```

The "pnorm" output:

```
                  Facet Normals and Offsets
                  -------------------------

               Hull Dimension=4 Number Facets=12
```

```
--------------------------------
```

1	-0.09198	0.6762	0.6528	-0.3289	-0.07685
2	-0.5268	-0.5448	-0.5395	-0.3669	0.02191
3	-0.807	-0.3889	-0.2425	-0.3724	-0.07446
4	-0.33	-0.4124	-0.588	-0.6126	0.09421
5	0.8529	-0.2789	-0.3865	-0.2131	-0.08718
6	0.734	-0.4798	-0.3216	-0.3572	0.04806
7	0.2045	0.7804	0.2935	-0.5128	-0.02724
8	0.2079	0.6345	0.3198	-0.6722	0.06815
9	-0.4132	0.2585	-0.8411	-0.2344	-0.2987
10	-0.05246	0.9981	-0.03052	-0.00885	-0.3634
11	0.07573	0.9324	-0.3512	-0.04	-0.4231
12	-0.006787	0.9799	-0.1457	-0.1361	-0.3287

The "pinci" output (similar "pvert"):

```
Facet Incidences: Number of Facets=12
-------------------------------------

 1  4 : 5 0 7 2
 2  4 : 7 5 1 8
 3  4 : 0 5 7 8
 4  4 : 5 4 7 1
 5  4 : 4 1 3 2
 6  4 : 4 7 1 2
 7  4 : 5 4 3 2
 8  4 : 4 5 7 2
 9  4 : 6 4 5 1
10  4 : 5 6 9 3
11  4 : 6 4 9 3
12  4 : 4 6 5 3
```

The "ptria" output:

```
Centers: Number Facets=12
-------------------------


 1   1.11e-016  1.943e-016  2.776e-016
 2 -1.665e-016 -1.665e-016 -1.756e-016
 3  -2.22e-016  -1.11e-016 -8.218e-017
 4           0  -1.11e-016  -1.11e-016
 5  3.886e-016  -2.22e-016  -2.22e-016
 6  3.331e-016 -1.665e-016           0
```

```
 7 -5.551e-017 -3.331e-016 -5.551e-017
 8           0  1.665e-016  8.327e-017
 9 -2.776e-016  1.665e-016 -5.551e-016
10 -6.106e-016  1.193e-014 -2.776e-016
11  5.551e-017  1.776e-015 -6.106e-016
12 -1.665e-016  1.055e-015 -3.608e-016
```

The "pvneig" output:

```
Vertex Neighbors: Number Points 10
----------------------------------

 6 : -27 -21 -14  -7   0   2
12 : -39 -38 -36 -31 -13 -12  -3   1   3   4
       5   8
12 : -21 -20 -14 -13 -12  -7  -3   0   4   5
       6   7
 8 : -31 -20 -13   4   6   9  10  11
10 : -36 -31   3   4   5   6   7   8  10  11
14 : -40 -39 -27 -21 -20   0   1   2   3   6
       7   8   9  11
 8 : -40 -39 -38 -36   8   9  10  11
 8 :  -7  -3   0   1   2   3   5   7
10 : -40 -39 -38 -27 -14 -12  -7  -3   1   2
12 : -40 -38 -36 -31 -27 -21 -20 -14 -13 -12
       9  10
```

The "pextr" output:

```
Vertex Extremes: Number of Points 10
------------------------------------

          5 0 7 2 1 8 4 3 6 9


        QHULL invoked by options:

 Pfacets  Pincidence  Pnormals  Ppoints  Psummary  Pneighbors
PNeighbors-vertex  PVertex-average  PExtremes  Pstatistics
QbBound-unit-box  Qcoplanar-keep  Farea  _pre-merge  _zero-centrum
Pgood  Qinterior-keep  _max-width 0.88  Error-roundoff 8.9e-016
_one-merge 8e-015  Visible-distance 5.3e-015  U-coplanar-distance
5.3e-015  Width-outside 1.1e-014  _wide-facet 3.2e-014


        Precision constants:
```

```
    0.88 max. abs. coordinate in the (transformed) input ('Qbd:n')
8.9e-016 max. roundoff error for distance computation ('En')
4.5e-016 max. roundoff error for angle computations
1.1e-014 min. distance for outside points ('Wn')
5.3e-015 min. distance for visible facets ('Vn')
5.3e-015 max. distance for coplanar facets ('Un')
3.2e-014 max. facet width for recomputing centrum and area
       1 max. cosine for pre-merge angle
1.8e-015 radius of pre-merge centrum
       1 max. cosine for post-merge angle
  8e-015 max. distance for merging two simplicial facets
1.1e-016 max. roundoff error for arithmetic operations
  2e-308 min. denominator for divisions zero diagonal for Gauss:
 4.14e-015 7.48e-015 1.17e-014 1.96e-014
```

The "pstat" output:

```
                      Memory statistics:

     120 quick allocations
     156 short allocations
       8 long allocations
     129 short frees
       8 long frees
    9424 bytes of short memory in use
     464 bytes of short memory in freelists
    1644 bytes of long memory allocated (except for input)
       0 bytes of long memory in use (in 0 pieces)
   65536 bytes per memory buffer (initially 131072 bytes)
      14 calls to qh_setlarger
     6.7 average copy size


                    Freelists (bytes->count):
 24->2 32->3 48->3 176->1


  Size in Bytes: merge 24 ridge 28 vertex 44 facet 176 normal 32
         Ridge Vertices 20 Facet Vertices or Neighbors 24


  print "GOF=",gof;
  print "Vertices=",vert;
  print "Offset=",offs;
  print "Centers=",cent;
  print "Neighbors=",neigh;
```

GOF=

	1
Failure	0.00000
Time	0.00000
NFacets	25.000
NVertices	10.0000
Area	0.12205
Volume	0.00000
unused	.
	.
	.
	.

Vertices=

	Vertex0	Vertex1	Vertex2	Vertex3
Fac_0	5	0	7	2
Fac_1	5	7	1	8
Fac_2	5	0	7	8
Fac_3	4	5	7	1
Fac_4	4	1	3	2
Fac_5	4	7	1	2
Fac_6	4	5	3	2
Fac_7	4	5	7	2
Fac_8	6	4	5	1
Fac_9	6	5	9	3
Fac10	6	4	9	3
Fac11	6	4	5	3

Offset=

	Offset	Inner	Area
Fac_0	-0.07685	-1e-015	0.01634
Fac_1	0.02191	-1e-015	0.02835
Fac_2	-0.07446	-9e-016	0.01273
Fac_3	0.09421	-1e-015	0.01474
Fac_4	-0.08718	-9e-016	0.00180
Fac_5	0.04806	-9e-016	0.01222
Fac_6	-0.02724	-9e-016	0.00625
Fac_7	0.06815	-9e-016	0.02759
Fac_8	-0.29873	-1e-015	0.00106

Fac_9	-0.36342	-9e-016	0.00006
Fac10	-0.42310	-9e-016	0.00016
Fac11	-0.32875	-9e-016	0.00076

Centers=

	DIM_1	DIM_2	DIM_3
Fac_0	1.110e-016	1.943e-016	2.776e-016
Fac_1	-1.665e-016	-1.665e-016	-1.756e-016
Fac_2	-2.220e-016	-1.110e-016	-8.218e-017
Fac_3	0.000e+000	-1.110e-016	-1.110e-016
Fac_4	3.886e-016	-2.220e-016	-2.220e-016
Fac_5	3.331e-016	-1.665e-016	0.000e+000
Fac_6	-5.551e-017	-3.331e-016	-5.551e-017
Fac_7	0.000e+000	1.665e-016	8.327e-017
Fac_8	-2.776e-016	1.665e-016	-5.551e-016
Fac_9	-6.106e-016	1.193e-014	-2.776e-016
Fac10	5.551e-017	1.776e-015	-6.106e-016
Fac11	-1.665e-016	1.055e-015	-3.608e-016

Neighbors=

	Fac_0	Fac_1	Fac_2	Fac_3
Fac_0	-7	7	-21	2
Fac_1	-3	-39	2	3
Fac_2	-7	1	-27	0
Fac_3	1	5	8	7
Fac_4	-13	6	5	-31
Fac_5	-3	4	7	3
Fac_6	-20	4	7	11
Fac_7	0	5	6	3
Fac_8	3	-39	-36	11
Fac_9	-20	10	11	-40
Fac10	-31	9	11	-36
Fac11	6	9	10	8

6 Illustrations

6.1 On Testing the Distribution of Sample Data

For all tests the sample data can be (univariate) vectors or matrices where the columns define the variates and the rows define the sample observations.

`adtest()` there are different versions of this test implemented for testing the univariate distribution of the input data for uniformity or normality.

- When testing the data for uniformity, it is assumed that the data are in $[0, 1]$.
- When testing data for univariate normality, the mean and standard deviation of the sample data are being used for standardizing the data into $N(\mu = 0, \sigma = 1)$ form.

`jarbera()` testing the data for univariate normality. The mean and standard deviation of the sample data are being used for standardizing the data into $N(\mu = 0, \sigma = 1)$ form.

`shapwilk()` testing the data for normality. The mean and standard deviation of the sample data are being used for standardizing the data into $N(\mu = 0, \sigma = 1)$ form.

`kstest()` there are different versions of this test implemented which can be specified with `optn[3]`:

0 the default "cdf" version: The string input argument `dist` can be used to specify one of a larger set of typical distributions. There may be a variable number of additional input arguments specifying non-default distributional parameters. Note, if no additional parameters are specified e.g. with the `"norm"` distribution, the distribution of the sample data w.r.t. the standard normal distribution (i.e. with zero mean und unit standard deviation) is being tested.

1 Anderson-Darling approximative for testing univariate uniformity: the data are assumed to be in $[0, 1]$ otherwise they will be normalized using location and scale of the sample

2 Marsaglia-Tsang-Wang exact (JSS,2003) for testing univariate uniformity: the data are assumed to be in $[0, 1]$ otherwise they will be normalized using location and scale of the sample

3 an older version of the KS-Lilliefors method for testing univariate normality: the data are assumed to be standardized with $\mu = 0$ and $\sigma = 1$, otherwise they will be standardized to zero mean and unit standard deviation using the corresponding sample moments

4 Alan Miller's code for the KS-Lilliefors method for testing univariate normality: the data are assumed to be standardized with $\mu = 0$ and $\sigma = 1$, otherwise they will be standardized to zero mean and unit standard deviation using the corresponding sample moments

`berkow()` there is only one version implemented which corresponds to the default ("cdf") version of the `kstest()` function (see `kstest()`).

`mvntest()` this function implements the following seven tests for multivariate normality where the sample data are in matrix form with columns defining the variates and the rows the observations:

- Mardia's tests for multivariate skewness
- Mardia's tests for multivariate kurtosis
- Mardia & Foster (1983) omnibus test for skewness and kurtosis
- Royston (1983) W test (extension of Shapiro-Wilks test)
- $W_{min}(5)$ test by Wang and Hwang (2011)
- Henze-Zirkler (1990) test
- Small's $Q1$, $Q2$, $Q3$

Most of these tests use the sample mean vector and covariance matrix for applying a test to "standardized" data.

The "cdf" versions of the `kstest()` and `berkow()` functions may be more difficult (and error prone) for the user but can be much more general than the `adtest()`, `jarbera()`, and `shapwilk()` functions which automatically use the sample mean and standard deviation for standardizing the input data before they are being tested w.r.t. the standard normal distribution. However, there would be no problem to extend the `adtest()`, `jarbera()`, and `shapwilk()` functions to use other means and standard deviations than those computed from the input sample.