CMAT Newsletter: December 2007

Wolfgang M. Hartmann

December 2007

Contents

1 General Remarks

The old function `dim` was renamed to `size` to be consistent with Matlab. The new function `dim` returns the (integer) number of dimensions of a data object.

1.1 New Functions

The following new functions are implemented:

dim returns an integer which is the number of dimensions of an data object, i.e. for scalars 0, for vectors 1, for matrices 2, etc. (The old `dim` function was renamed to `size`)

irtml Maximum Likelihood estimation of the Samejima graded and Rasch parametric IRT scaling models

irtms returns scales and Loevinger H coefficients for (nonparametric) IRT Mokken scaling

mds computes a variety of methods for *metric* and *nonmetric* multidimensional scaling (MDS) for 2-and 3-way data tables

setdiff set difference of `a` and `b`

setisect set intersection of `a` and `b`

setmembr binary membership of `a` in `b`

setunion set union of `a` and `b`

setxor set XOR of `a` and `b`

size returns the size of dimensions of a data object like vector, matrix, tensor, list. (This is the old `dim` function.)

sortrow sorts the rows of a matrix w.r.t. a sorting key based on a (sub)set of the columns of that matrix; applied to categorical variables (int columns) this function is useful also for obtaining multidimensional frequency tables

unique returns unique members of vector

vec2tri moves entries of vector to lower or upper triangular or symmeyric matrix

Note, that the MDS function is still in the works. Only versions of the COSPA, SUMSCAL, INDSCAL, and KYST programs are included. The next newsletter will report about MULTISCALE and SMACOF extensions.

1.2 Fixed Bugs

Many bugs were fixed, especially some with tensors and data lists. Some operations now also work with string and mixed data type (numeric and string) tensors.

2 Modifications of Features

2.1 Modification of `qp` Function

Until release 4 the old call of the `qp` function
$<$ `xr,rp` $>$ `=` `qp(`$sym, vec <, x0 <, optn >>$`)`
was not compatible with the specification of the `lp` function
$<$ `xr,lm,rp` $>$ `=` `lp(`"$meth$", $c, lau <, optn <, lubc >>$`)`
With the old `qp` call, linear (lau) and boundary ($lbub$) constraints had to be specified inside the `optn` argument. To make the use of the two function easier, the order and definition of input arguments for the `qp` function was made compatible with that of the `lp` function, and was been changed to the new call:
$<$ `xr,rp` $>$ `=` `qp(`$sym, vec <, lau <, optn <, x0 <, lbub >>>>$`)`
The definition of the input arguments `lau` and `lbub` is the same as for the `lp` function:

lau is the name of a $n_{lc} \times n + 2$ or $n_{lc} \times n + 1$ matrix with lower bounds (first column), coefficients, and upper bounds (last column) for n_{lc} general linear constraints. The specification of missing values for lower or upper bounds means that the corresponding side constraint is not imposed. If `lau` contains only $n+1$ columns, where n is defined by the dimension of the first argument `c`, then no upper bounds are imposed. This argument may be specified as a missing value if only boundary constraints are imposed with the last input argument.

lubc (optional) is the name of a $n \times 2$ matrix with lower (first column) and upper (second column) simple boundary constraints.

When changing the input scripts of the test examples, the objects specifying linear and boundary constraints only had to be moved from the `optn` argument into the call command. Nothing else was changed for the `qp` function. The `lp` function remains the same.

2.2 Modification of `nlp` Function

To be compatible with the `lp` and the new `qp` functions the input arguments of the `nlp` function were extended with linear and boundar constraints. Until release 4 the old call of the `nlp` function
$<$ `xr,rp` $>$ `=` `nlp(`$func, x0 <, optn <, nlcon, < grad, < hess, < jcon >>>>>$`)`

With the old `nlp` call, linear (*lau*) and boundary (*lbub*) constraints had to be specified inside the `optn` argument. To make the use of the two function easier, the order and definition of input arguments for the `nlp` function was made more compatible with that of the `lp` and `qp` functions, and was been changed to the new call:
$<$ `xr,rp` $>$ `= nlp(`$func, x0 <, optn <, lau <, lbub <, nlcon, < grad, < hess, < jcon >>>>>$`)`
The definition of the input arguments `lau` and `lbub` is the same as for the `lp` and the `qp` functions:

lau is the name of a $n_{lc} \times n + 2$ or $n_{lc} \times n + 1$ matrix with lower bounds (first column), coefficients, and upper bounds (last column) for n_{lc} general linear constraints. The specification of missing values for lower or upper bounds means that the corresponding side constraint is not imposed. If `lau` contains only $n+1$ columns, where n is defined by the dimension of the first argument `c`, then no upper bounds are imposed. This argument may be specified as a missing value if only boundary constraints are imposed with the last input argument.

lubc (optional) is the name of a $n \times 2$ matrix with lower (first column) and upper (second column) simple boundary constraints.

When changing the input scripts of the test examples, the objects specifying linear and boundary constraints only had to be moved from the `optn` argument into the call command. Nothing else was changed for the `nlp` function.

2.3 Extensions to Various Functions

2.3.1 Extensions to `svm()` Function

The linear chunking algorithm for linear L1 SVM was implemented, see Mangasarian & Thompson (2006a): "Massive data classification via unconstrained support vector machines". This method is specified with the `"meth" "l1lin"` option together with the new `nchunk` option which specifies in how many chunks a data set is decomposed.
Chunking can be necessary when a data set is given which has a huge number of rows (observations) but not so many columns (variables).
Using the new `chunk` option the user can specify one of three methods for decomposing the data set in chunks:

"chunk" "blo" blockwise decomposition: the data set is sequentially divided in k chunks each with approximately $Nobs/k$ observations; this is the default and agrees with that used by Mangasarian & Thompson (2006)

"chunk" "spl" splitwise decomposition: Usually split sampling will be better than block sampling which is sensitive to ordered training data.

"chunk" "ran" random decomposition: the data set is randomly decomposed in k chunks each with approximately $Nobs/k$ observations.

These methods correspond to the k-fold cross classification methods with the same names.
The data set `heart_scale` has only 270 observations an 13 predictor variables and is not a typical case for this algorithm. Usually data sets with many more observations could be tackled with the chunking method. A nonlinear SVM would be more appropriate for these data.
The following example uses blockwise chunking and does not use the parameter tuning feature for the (α, δ) arguments and fixes $\alpha = 1$ and $\delta = .001$ as by Thompson and Mangasarian.

```
data = rspfile("..\\tdata\\heart_scale.dat");
modl = "1 = 2:14";
class = 1;

/*--- L1LIN chunking: with direct Cholesky: 4 chunks ---*/
optn = [ "print"           2 ,
         "popt"            2 ,
         "pplan"             ,
         "meth"      "l1lin" ,
         "lines"             ,
         "nchunk"          4 ,
         "peneps"       .001 ,
         "dbeg"        0.001 ,
         "dend"        0.001 ,
         "abeg"        1. ,
         "aend"        1. ,
         "c"              1. ,
         "kern"       "line" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class);

                      *****************
                      Model Information
                      *****************

                Number Valid Observations  270
                Response Variable          Y[1]
                N Independend Variables     13
                Support Vector C Classification
                Model with 1 Linear Constraint
                Estimation Method
                Kernel Function         Linear
                Use Unscaled Predictor
                Bias Corrected Predicted Values
```

****
Class Level Information
****

Class	Level	Value
Y[1]	2	-1 1

Traindata stored incore: Mem need: 0.027 Mem spec: 2 Mb

****
Number of Observations for Class Levels
****

Variable	Value	Nobs	Proportion
Y[1]	-1	150	55.555556
	1	120	44.444444

L1LIN Linear Chunking History (alpha=1 delta=0.001)

Iter	Subst	ChunkS	Nconst	Niter	Gnrm	Crit	CritRatio
1	1	67	0	160	1.002e-007	17.9537749	1.0000000
2	2	91	23	119	6.316e-007	38.8264144	1.0000000
3	3	120	53	43	3.340e-007	65.6342394	1.0000000
4	4	138	70	95	5.187e-007	85.7231269	1.0000000
5	1	144	77	198	3.932e-007	87.9461728	0.6609295
6	2	143	75	55	5.933e-007	89.6480619	0.3955778
7	3	141	74	99	8.370e-007	90.2314959	0.1578105
8	4	145	77	87	7.682e-007	89.5075104	0.0215966
9	1	143	76	44	8.036e-007	89.2065499	0.0071146
10	2	140	72	309	5.057e-007	89.1662683	0.0026944
11	3	142	75	89	4.810e-007	89.8611906	0.0020562
12	4	146	78	277	5.801e-007	89.4371936	3.93e-004
13	1	143	76	61	6.443e-007	90.0562587	0.0047400
14	2	139	71	22	5.228e-007	89.1850281	1.05e-004
15	3	145	78	123	8.522e-007	90.0479764	0.0010382
16	4	145	77	38	3.353e-007	91.8281751	0.0131905
17	1	147	80	29	7.125e-007	90.9067039	0.0046996
18	2	143	75	39	9.067e-007	90.7068797	0.0084598
19	3	143	76	68	9.138e-008	90.7662633	0.0039725
20	4	144	76	239	4.189e-007	90.6424077	0.0064984
21	1	146	79	63	6.881e-007	90.1361614	0.0042561
22	2	142	74	42	7.128e-007	90.3350137	0.0020540
23	3	146	79	213	9.015e-007	90.2562454	0.0028174
24	4	143	75	47	5.925e-007	90.8644292	0.0012232

```
   25     1    143     76     80 4.509e-007 90.3824314  0.0013642
   26     2    144     76     83 5.209e-007 91.3074641  0.0053537
   27     3    139     72    198 1.083e-007 89.3718416  0.0049235
   28     4    143     75    112 7.809e-007 89.1628497  0.0094518

     Delta=0.001 Alpha=1 Criterion=89.1628 Misclassified=41
C=1 : Solution for delta=0.001 alpha=1 selected based on Training
                            fit.

         Linear Separating Plane (w*x = 1.21385)
         ***************************************

               Dense Row Vector (ncol=14)

R |           1          2          3          4          5
      0.0000000  0.3310847  0.4181960  0.2646343  0.8525023

R |           6          7          8          9         10
     -0.1858829  0.2428280 -0.6553894  0.2833240  0.4920461

R |          11         12         13         14
      0.2018449  1.0462386  0.5063975  1.2138462

              Largest 12 Plane Coefficients (Sorted)
              **************************************

                      1 12  X13  1.046238625
                      2  5  X6   0.852502260
                      3  8  X9  -0.655389417
                      4 13  X14  0.506397494
                      5 10  X11  0.492046108
                      6  3  X4   0.418195960
                      7  2  X3   0.331084748
                      8  9  X10  0.283323955
                      9  4  X5   0.264634285
                     10  7  X8   0.242828046
                     11 11  X12  0.201844877
                     12  6  X7  -0.185882871
               Sparsity: 12 Nonzeros 1 Zeros (C=1)

                 *******************************
                 Evaluation of Training Data Fit
                 *******************************

     Index                              Value       StdErr
     Absolute Classification Error         41        .
     Classification Accuracy        84.81481481       .
```

```
Concordant Pairs              71.13333333      .
Discordant Pairs              2.300000000      .
Tied Pairs                    26.56666667      .
Goodman-Kruskal Gamma         0.937358148  0.020748504
Kendall Tau_a                 0.341181330      .
Kendall Tau_b                 0.691703996  0.044255926
Stuart Tau_c                  0.679835391  0.045074591
Somers D C|R                  0.688333333  0.044666408
Somers D R|C                  0.695091164  0.044309177
```

Classification Table

Observed |	Predicted -1	Predicted 1
-1	132	18
1	23	97

```
Regularization Parameter C . . . . . 1
Kernel Function. . . . . Linear
Norm of Longest Vector . . . . . 3.28753
Number Misclassifactions (Training Data) . . . . . 41
Number Fast Runs through TrainData . . . . . 13664
Number Slow Runs through TrainData . . . . . 10635
Total Number of Kernel Calls . . . . . 37666
Time for Optimization. . . . . 30
Total Processing Time. . . . . 30
Optimization Criterion . . . . . 89.1628
Infinity Norm of Gradient. . . . . 5.77993e-007
Geometric Margin . . . . . 1.09789 (|w|^2= 3.31853)
Number Support Vectors . . . . . 270 (100.00 %)
Number Support Vectors on Margin . . . . . 90
Bias . . . . . -0.0244385
Radius of Sphere Around SV . . . . . 3.28753
Estimated VCdim of Classifier. . . . . 36.8662
Linear Kernel Constant (Fit) . . . . . 1.21385
Linear Kernel Constant (PCE) . . . . . .
```

Linear Separating Plane (w*x = 1.21385)

Dense Row Vector (ncol=14)

R |	1	2	3	4	5
	0.0000000	0.3310847	0.4181960	0.2646343	0.8525023

```
R |           6          7          8          9         10
     -0.1858829  0.2428280 -0.6553894  0.2833240  0.4920461

R |          11         12         13         14
      0.2018449  1.0462386  0.5063975  1.2138462

              Largest 12 Plane Coefficients (Sorted)
              **************************************

                       1 12  X13  1.046238625
                       2  5  X6   0.852502260
                       3  8  X9  -0.655389417
                       4 13  X14  0.506397494
                       5 10  X11  0.492046108
                       6  3  X4   0.418195960
                       7  2  X3   0.331084748
                       8  9  X10  0.283323955
                       9  4  X5   0.264634285
                      10  7  X8   0.242828046
                      11 11  X12  0.201844877
                      12  6  X7  -0.185882871
               Sparsity: 12 Nonzeros 1 Zeros (C=1)

               Total Number of Kernel Calls: 37666
                    Time for Optimization: 30
                    Total Processing Time: 30
```

Running the same example with the L1FSM method (the equivalent method without chunking) we specify:

```
optn = [ "print"           2 ,
         "popt"            2 ,
         "pplan"              ,
         "meth"      "l1fsm" ,
         "lines"              ,
         "peneps"       .001 ,
         "dbeg"        0.001 ,
         "dend"        0.001 ,
         "abeg"           1. ,
         "aend"           1. ,
         "c"              1. ,
         "kern"       "line" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class);
```

We now obtain only 40 misclassifications:

```
*******************************
Evaluation of Training Data Fit
*******************************

Index                              Value       StdErr
Absolute Classification Error         40            .
Classification Accuracy        85.18518519         .
Concordant Pairs               72.05000000         .
Discordant Pairs               2.216666667         .
Tied Pairs                     25.73333333         .
Goodman-Kruskal Gamma          0.940305206  0.019892544
Kendall Tau_a                  0.346137960         .
Kendall Tau_b                  0.699578135  0.043801392
Stuart Tau_c                   0.689711934  0.044509315
Somers D C|R                   0.698333333  0.044052315
Somers D R|C                   0.700825156  0.043911181

                Classification Table
                --------------------

                        |    Predicted
                Observed |       -1       1
                ---------|-----------------
                      -1 |      131      19
                       1 |       21      99
```

And the direct method is much faster:

```
Total Number of Kernel Calls: 37666
     Time for Optimization: 0
     Total Processing Time: 1
```

The corresponding CMAT listings for the chunking method are the following:

```
/* CMAT Formulation of LP_SVM Algorithm:
   similar to tsvm18.inp: for Nobs > nvar: with SMW */

function lpsvchnk(A,y,nu,delta,alf,eps1,armlin) {
 m = nrow(A); n = ncol(A); np1 = n + 1;
 rnu = 1. / nu; tol = 1.e-6; maxi = 1000;
 ubnd = -.01;
 epsm = cons(m,1,eps1); num = cons(m,1,nu);
 /* print "Size A:", size(A);
    print "Size y:", size(y); */

 /* Starting u[m] */
 u = cons(m,1,1.); umin = 1.;
```

```
D = diag(y); DA = D * A;

/* Prepare F and G */
du = D * u; Adu = A‘ * du;
pp = max(Adu-1,0.); np = max(-Adu-1.,0.);
dsum = du[+]; dd = dsum * y;
unu = max(u-num,0.); uu = max(-u,0.);

/* Function */
t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf*uu‘*uu;
t3 = dsum * dsum;
crit = -eps1*u[+] + .5 * (t1 + t3 + t2);
/* print "t1,t2,t3,crit=",t1,t2,t3,crit; */

/* Gradient */
t1 = y .* (A*pp); t2 = y .* (A*np);
grd = -epsm + t1 - t2 + dd + unu - alf * uu;
gnrm = norm(grd);
/* print "gnrm=",gnrm," grad=",grd; */

iter = 0;
while (++iter <= maxi && (gnrm > tol || umin <= ubnd)) {
  uold = u; cold = crit;
  /* Hessian */
  e = sqrt(sign(np) + sign(pp));
  E = diag(e);
  B = DA * E; H = [ B y ];
  /* that’s different */
  f = delta + sign(unu) + alf*sign(uu);
  f = 1. / f;
  F = diag(f);
  gg = f .* grd; b = H‘ * gg;
  C = H‘ * F * H + ide(np1);

  di = C \ b;
  di = f .* (H * di); di -= gg;
  u = uold + di;

  /* Prepare */
  du = D * u; Adu = A‘ * du;
  pp = max(Adu-1.,0.); np = max(-Adu-1.,0.);
  dsum = du[+]; dd = dsum * y;
  unu = max(u-num,0.); uu = max(-u,0.);

  /* Function */
  t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf * uu‘*uu;
```

```
    t3 =  dsum * dsum;
    crit = -eps1*u[+] + .5 * (t1 + t3 + t2);
    /* print "t1,t2,t3=",t1,t2,t3;
       print "crit=",crit," di=",di; */

    /* Armijo line search for beta */
    beta = 1.;
    if (armlin && crit >= cold) {
      beta = .5;
      while (beta > 1.e-12) {
        u = uold + beta * di;
        du = D * u; Adu = A` * du;
        pp = max(Adu-1,0.); np = max(-Adu-1.,0.);
        dsum = du[+]; dd = dsum * y;
        unu = max(u-num,0.); uu = max(-u,0.);
        /* Function */
        t1 = pp`*pp + np`*np; t2 = unu`*unu + alf * uu`*uu;
        crit = -eps1*u[+] + .5 * (t1 + dsum*dsum + t2);
        if (crit < cold) break;
        beta *= .5;
    } }

    /* Gradient */
    t1 = y .* (A*pp); t2 = y .* (A*np);
    grd = -epsm + t1 - t2 + dd + unu - alf * uu;
    gnrm = norm(grd); umin = min(u);
    /* print "Iter=",iter," Crit=",crit," Gnrm=",gnrm,
          " umin=",umin," Beta=",beta; */
    /* print "grad=",grd; */
  }

  /* final weights and intercept gamma */
  w = (pp - np) / eps1;
  gamma = -du[+] / eps1;
  yv = max(u-num,0) / eps1;
  vc = pp + np;
  crit = num` * yv + vc[+];
  print "  RESULT LPSVM: gamma=",gamma," Crit=",crit," Beta=",beta;
  print "  Niter=",iter," gnrm=",gnrm," umin=", umin;
  print "Coeff w=",w;
  /* print "Yv=",yv; */
  return(w,gamma,yv,crit);
 }
/*-------------------------------------------*/
 /* CMAT Formulation of ChunkingSVM Algorithm:
```

```
   by Michael Thompson and Olvi Mangasarian
   for Nobs > nvar: with SMW */

function SVMchunk(A,y,nu,delta,alf,eps1,nchk,prec) {
 armlin = 1;
 m = nrow(A); n = ncol(A); np1 = n + 1;
 rnu = 1. / nu; tol = .001; maxi = 1000;
 epsm = cons(m,1,eps1); num = cons(m,1,nu);

 k = nchk; indx = [ 0:k ]; rm = (real)m;
 indx = floor(rm * indx / k);
 print "INDX=", indx;

 ecnt = 0; cold = macon("mbig");
 iter = 0; ncon = 0;
 free cind,ind2;
 while (ecnt < 3) {
   ic = iter % nchk + 1;
   i1 = indx[ic]+1; i2 = indx[ic+1];
   ind1 = [ i1 : i2 ]‘;
   Ach = A[ind1,]; dch = y[ind1];
   if (ncon) {
     free ind2; k = 0;
     for (i = 1; i <= ncon; i++) {
       id = cind[i];
       if (id < i1 || id > i2) ind2[++k] = id;
     }
     if (k) {
       Ach = Ach |> A[ind2,];
       dch = dch |> y[ind2];
       ind1 = ind1 |> ind2;
       /* print "Ind1=",ind1; print "Ind2=", ind2; */
   } }
   mch = nrow(Ach);
   print "Iter=", iter," ncon=",ncon," size=",mch;

   < w,gamma,yv,crit > =
              lpsvchnk(Ach,dch,nu,delta,alf,eps1,armlin);

   ww = diag(dch) * (Ach * w - gamma);
   /* print "ww=",ww; */
   vcon = ww + yv - 1.;
   ncon = 0; free cind;
   for (i = 1; i <= mch; i++)
   if (vcon[i] < 1.e-2 || yv[i] > 0.) cind[++ncon] = ind1[i];
```

```
    dm = 1. / max(cold,crit);
    pct = dm * abs(cold - crit);
    print "  End Outer Iter: Pct=",pct," new ncon=",ncon;
    if (pct < prec) ecnt++;
    else { ecnt = 0; cold = crit; }
    iter++; if (iter == maxi) break;
   }
   return(w,gamma);
  }

/*-------------------------------------------*/

  print "Problem HEART from Statlog collection";
  print "Michie, Spiegelhalter, & Taylor, 1994";
  data = rspfile("..\\tdata\\heart_scale.dat");

  x = data[,2:14]; y = data[,1];
  eps1 = .001; nu = 1.; nchk = 4;
  alfa = 1.; delta = .001; /* default setting */
  prec = .001;
  print " CALL of SVMchunk: prec=", prec, " nchk=",nchk;
  print " delta=", delta," alfa=", alfa;
  < w, gamma > = SVMchunk(x,y,nu,delta,alfa,eps1,nchk,prec);

  print "SVMchunk Result: Gamma=", gamma;
  print "SVMchunk Result: weights=", w;
```

3 New Developments

3.1 Function `dim`

nd = dim(a)

Purpose: The function `dim` returns an integer which is the number of dimensions of the data object `a`.

Input: The only input argument must be a adata object `a`.

Output: An integer is returned. For

1. scalars, `nd=0`
2. vectors, `nd=1`
3. matrices, `nd=2`
4. tensors, `nd=i` where $i > 2$

is returned.

Restrictions: There are no obvious restrictions for the input object `a`.

Relationships: nrow(), ncol(), size()

Examples: 1. Scalar, Vector, and Matrix:

```
brv = [ 1 2 3 ];                          Dim of row vector=1
dbr = dim(brv);
print "Dim of row vector=",dbr;           Dim of col vector=1

bcv = [ 1, 2, 3 ];                        Dim of matrix=2
dbc = dim(bcv);
print "Dim of col vector=",dbc;

bmat = [ 100. 200. 300.,
         300. 200. 100. ];
db2 = dim(bmat);
print "Dim of matrix=",db2;
```

2. Tensor:

```
real B1[3,4,5];                           Dimension B1=3
print "B1[3,4,5]=", B1;
n1 = dim(B1);
print "Dimension B1=", n1;
```

3.2 Function `irtml`

< gof,est,cov,resi,stmres,sigmar,m2tab > = irtml(data,optn<,tini<,bini<,bc<theta>>>>)

Purpose: For entirely categorical data $\mathbf{X}$ with N rows (observations) and n items where item j has m_j categories (levels) and all items have a total of $m = \sum_j^n m_j$ categories, this function fits one of three parametric IRT models:

- the *Samejima* parametrized model containing of $\tilde{m} = \sum_j^n (m_j - 1)$ thresholds for the levels $1, \ldots, m_j - 1$ and n slopes b_j, $j = 1, \ldots, n$. This model has $npar = m$ parameters to estimate.
- the *Rasch* parametrized model of $\sum_j^n (m_j - 1)$ thresholds for the levels $1, \ldots, m_j - 1$ and one slope b. This model has $npar = m - n + 1$ parameters to estimate.
- The nominal scaling model by Bock (1972).

Note, it is not assumed that all variables have the same number of categories. The `nlp` optimization subroutines are used to estimate the parameters for the three models. In addition the EM and the BCL algorithms can be used for fitting the Samejima graded model. Using the `noopt` option, fit statistics, asymptotic standard errors, confidence intervals (except profile likelihood CIs, requiring optimization), and (raw and standardized marginal) residuals can be computed, printed, and returned for the original input estimates `tini` and `bini`.

The $X2$ and $M2$ fit statistics (Maydeu-Olivares & Joe, 2005) are computed for the two ordinal scaling models. Confidence intervals (CIs) for the parameter estimates are computed:

- Wald confidence intervals based on the inverse of the Hessian matrix,
- ASEs and CIs based on the Godambe matrix,
- Profile Likelihood CIs.

Note, some of the statistics for ML estimation of IRT models assumes that the data have distinctive patterns. That usually requires a frequency column in the data. If the data does not have a frequency column, the data will be preprocessed by default and reduced to distinctive patterns. (However, this preprocessing may be avoided by specifying the `nofad` option.) The same data could be generated by running the `sortrow` function.

Note, since the data may include many zero values, in some applications sparse storage of the data is used.

For nominal scaling (Bock, 1972) there are two applications:

1. The modeling (training) approach: requires a data set with an ability variable, i.e. an individual ability score.

2. The scoring appraoach: Using the `noopt` option and specifying the `theta` input argument, the observations of a data can be scored, without recomputing model estimates.

The maximum number of categories of all items must be specified with the `nrcat` option. There may be items which have categories with zero frequencies.
This function is based on the excellent Matlab and C code written by Harry Joe.

Note, some of the features may be computationally too expensive (need too much memory or too much computer time) for many variable n, many categories m, or many parameters $npar$:

1. The Fisher scoring matrix is used for computing the Godambe matrix, the $\mathbf{\Sigma_r}$ matrix, and the standardized residuals. For the Fisher scoring matrix the first derivatives w.r.t. all $\prod_j^n m_j$ patterns is needed which could be a very large number and computationally not feasible.
2. The Fisher scoring matrix and related statistics should not be computed for $\prod_j^n m_j \geq 2^{31}$. For example, for $n = 20$ variables with $m = 4$ categories we obtain more than 10^{12} possible patterns and only super computers would be able to compute the Fisher matrix.
3. The $M2$ statistics is default but may be suppressed with the `nom2` option.
4. The number of residuals of all patterns is the very large number $\prod_j^n m_j$. That return object may take too much memory and can be suppressed by specifying the `nores` option.
5. The profile likelihood confidence intervals may need too much computer time since it requires $2 * npar$ additional optimizations.

Input: data This is

- either an $N \times n$ matrix containing numerical categorical data,
- or $N \times (n+1)$ matrix with n columns containing numerical categorical data and an additional column containing individual frequencies,
- or only when the `"size"` option is specified, a $N \times (n+1)$ matrix with n columns containing numerical categorical data and an additional column containing individual probability values ($0 \leq p \leq 1$),
- and only for nominal scaling (Bock, 1972): an $N \times (n+1)$ or $N \times (n+1)$ matrix with n columns containing numerical categorical data, an additional column containing individual ability assesments (levels of a categorical variable), and optional, an additional column containing individual frequencies.

That means, we have the data

- either in *individial form*: one row per subject and no additional frequency column (frequency one is assumed, multiple patterns are permitted)
- or in *pattern form*: a frequency column is attached where the input of zero frequencies is permitted, but distinct patterns are assumed.

That means each data column, except the frequency column, should have integer values in the range $0, \ldots, m_j - 1$. The input of zero and non-integer frequencies is permitted. If the data does not contain a frequency column a preprocessing step is performed by default generating a reduced data set of sorted distinct patterns with a frequency column added. This default preprocessing can be skipped when the user specifies the `nofad` option. However, users should be aware that the test statistics may be invalid if the data contains mutiple patterns.

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.

tini This argument should be a matrix or vector of $\tilde{m} = \sum_j^n (m_j - 1)$ values start values for the threshold estimates. If this argument *tini* is missing, starting values for thresholds are generated automatically.

bini This argument should be either a vector of n values (for the Samejima parametrized ML model) or a scalar for the Rasch model defining the start value(s) for the slope(s).

bc This argument should be either

- a $npar \times 2$ matrix specifying lower boundary constraints in its first column and upper boundary constraints in its second column;
- or a $npar$ vector of lower bounds for all parameters.

The default setting of lower bounds is -10. for the thresholds and unlimited for the slopes. The default setting of upper bounds is 10. for all parameters.

Option	Second Column	Meaning
"abil"	int	number of data column with ability categories (only for nominal method)
"alfa"	real	α value for ASE's and CI's
"cl"	string	"wald": normal theory Wald CI's: based on diagonal of inverse Hessian "goda": based on diagonal of Godambe matrix "plik": profile likelihood CIs: each value is result of one optimization "none": no CI's
"emtol"	real	parameter change tolerance for EM algorithm
"emitr"	int	maximum number of iterations of EM algorithm
"fishsc"		computes and prints the Fisher scoring matrix (if the number of bivariate moments is not too large)
"freq"	int	specifies data column with frequency values
"inicon"		multiplies computed initial values for threshold with the constant 1.97403
"intpnt"	int	number points used for quadrature (integration)
"m2tab"	int	specifies the number q, $2 \leq q < n$ of variables of all subsets of the n input variables for which $M2$ values are computed
"method"	string	"sam": Samejima parametrized model "rasch": Rasch parametrized model "em": EM algorithm for Samejima "bcl": BCL algorithm for Samejima "nom": specifies Bock (1972) method for nominal scales including individual ability estimation
"nofad"		do not add a freqency column
"nom2"		do not compute $M2$
"nox2"		do not compute $X2$ and $M2$
"noopt"		do not optimize: input parameters must be provided
"nopr"		do not print any results
"nrabil"		Bock's original NR algorithm for scaling the individual ability (only for nominal method) (this is not recommended due to instability)
"nrcat"		number of categories (uopper limit of all items) (only for nominal method)
"orig"		Bock's original NR algorithm for estimating item parameters (works mostly)
"pcov"		print inverse Hessian matrix
"pdat"		print data matrix used in analysis
"pcov"		print inverse Hessian matrix and if specified with `cl` the Godambe matrix
"phis"	int	printed output of optimization history
"print"	int	specifies amount of printed output
"pres"		prints observationwise results
"ptab"		prints subset M2 table
"sing"	real	singularity criterion, default=1.e-8
"slope"	real	specify (initial) value for slope of Rasch model
"utfil"	int	specifies threshold for using utility file when computing $M2$
"wgt"	int	same as freq(uency) option

If profile likelihood or confidence intervals based on the Godambe matrix are specified with the `cl` option, the Wald ASEs and CIs are also computed (printed and returned in `est`. Note, that the $M2$ statistics, the Godambe matrix, and the Fisher scoring matrix can be computed only if the number of bivariate moments is not too large. In addition, most of the options of the `nlp` function can be specified for choosing an optimization technique and specifying crteria for terminating the iteration.

Output: There are at most seven return arguments:

gof is a vector of scalar results:

1. indicates error return
2. negative Likelihood value
3. determinant of Hessian matrix
4. $X2$ statistics
5. degrees-of-freedom of $X2$ statistics
6. $X2$ statistics
7. degrees-of-freedom of $X2$ statistics
8. maximum memory allocation (for testing)
9. largest amount of all memory allocated
10. total time in seconds
11. time used for parameter estimation in seconds
12. time for computing $X2$ and $M2$ in seconds

est is either

- a *npar* vector of parameter estimates;
- or a $npar \times 2$ matrix of parameter estimates in its first column and normal theory asymptotic standarad errors in its second column;
- or a $npar \times 4$ matrix of parameter estimates in its first column, normal theory asymptotic standarad errors in its second column, and lower and upper Wald confidence limits in its third and fourth column.
- if profile likelihood CIs are computed they are attached as additional columns

cov returns the $npar \times npar$ inverse Hessian matrix used for the covariance matrix. If the Fisher scoring matrix and/or the Godambe matrx is computed, a 3-way tensor is returned with either the two or three $npar \times npar$ matrices.

resi returns observed and predicted relative frequencies of the data patterns and its residual differences

stmres returns standardized marginal residuals based on the diagonal of the Σ_r matrix, where the first $\sum_{j=1}^{n}(m_j - 1$ values correspond to

univariate moments and the remaining $\sum_{j=1}^{n}\sum_{k=1}^{j}(m_j-1)(m_k-1)$ values correspond to bivariate moments.

sigmar returns the symmetric Σ_r matrix where the first $\sum_{j=1}^{n}(m_j-1$ rows and columns correspond to univariate moments and the remaining $\sum_{j=1}^{n}\sum_{k=1}^{j}(m_j-1)(m_k-1)$ rows and columns correspond to bivariate moments.

m2tab returns a table of subset $M2$ values

Restrictions: 1. Except for the frequency column the input data must be integer. For string data specifying categories the `replace` function could be used replacing strings by integers.

2. The data should not contain missing values. Either missing values are modeled as an additional level or observations with missing values should be removed from the analysis.

Relationships: irtms(), noharm()

Examples: 1. Samejima parametrized model for SIM3 Data: N=67, n=5, m=3

```
  sim3fr = [
#include "..\\tdata\\sim3fr.dat"
           ];
  sim3fr = shape(sim3fr,.,6);
  cnam = [ "v1":"v5" "wgt" ];
  sim3fr = cname(sim3fr,cnam);

  bini = [ 1.507444 2.616930 0.872721 0.463110 0.672905 ];
```

The initial values `tini` for thresholds are computed automatically:

```
  optn = [ "meth"     "same" ,
           "frequ"         6 ,
           "tech"   "nrridg" ,
           "phis"          3 ,
           "print"         3 ];
  < gof,est,cov > = irtml(sim3fr,optn,.,bini);


          ********************************************
          IRT ML Estimation (Maydeu-Olivares & Joe, 2006)
                      Fully Parametrized Model
          ********************************************

    Number Rows of Matrix . . . . . .          67
    Number Columns of Matrix  . . . . .         6
```

```
Number of Parameters. . . . . .          15
Total Number of Categories. . . . . .    15
Maximum Item Level. . . . . .             2
Column of Frequency Variable. . . . . .   6
Total Frequency . . . . . .             100
Number Uni- and Bivariate Moments . . . .  50
Number Integration Points . . . . . .     48
Integration Precision . . . . . .      1e-013
```

****
Frequencies of Data Categories
****

Variable	Level	Frequency	Percentage
v1	0	42	42.00 %
	1	13	13.00 %
	2	45	45.00 %
v2	0	38	38.00 %
	1	25	25.00 %
	2	37	37.00 %
v3	0	40	40.00 %
	1	16	16.00 %
	2	44	44.00 %
v4	0	39	39.00 %
	1	13	13.00 %
	2	48	48.00 %
v5	0	43	43.00 %
	1	23	23.00 %
	2	34	34.00 %

Initial Values for Thresholds

	1	2
v1	0.322773392	-0.200670695
v2	0.489548225	-0.532216814
v3	0.405465108	-0.241162057
v4	0.447312218	-0.080042708
v5	0.281851152	-0.663294217

Initial Values for Slopes

	1
v1	1.507444000
v2	2.616930000

```
v3   0.872721000
v4   0.463110000
v5   0.672905000


******************
Optimization Start
******************

Parameter Estimates
-------------------

Parameter      Estimate   Gradient   Lower BC   Upper BC

  1 v1_1     0.32277339 -7.8309638 -10.000000  10.000000
  2 v1_2    -0.20067070  7.7966570 -10.000000  10.000000
  3 v2_1     0.48954823 -13.905308 -10.000000  10.000000
  4 v2_2    -0.53221681  13.705799 -10.000000  10.000000
  5 v3_1     0.40546511 -3.9082620 -10.000000  10.000000
  6 v3_2    -0.24116206  3.7487666 -10.000000  10.000000
  7 v4_1     0.44731222 -1.2446625 -10.000000  10.000000
  8 v4_2    -0.08004271  1.2898429 -10.000000  10.000000
  9 v5_1     0.28185115 -2.5738892 -10.000000  10.000000
 10 v5_2    -0.66329422  2.6098443 -10.000000  10.000000
 11 Slope_1  1.50744400  0.2192942      .       10.000000
 12 Slope_2  2.61693000  3.3379225      .       10.000000
 13 Slope_3  0.87272100 -0.2063544      .       10.000000
 14 Slope_4  0.46311000 -0.3722770      .       10.000000
 15 Slope_5  0.67290500 -0.3011145      .       10.000000

          Value of Objective Function =       500.23


              Newton-Raphson Ridge Optimization
                  Without Parameter Scaling
                   User Specified Gradient
                User Specified Hessian (dense)


  Iteration Start:
  N. Variables             15
  N. Bound. Constr.        25        N. Mask  Constr.          0
  Criterion       500.2303533        Max Grad Entry  13.90530841
  N. Active Constraints     0


 Iter rest nfun act   optcrit  difcrit maxgrad   ridge      rho
```

```
  1    0    5   0  497.1282 3.102104 7.22164 0.50000  0.50392
  2    0    6   0  494.3578 2.770485 1.96759 0.25000  1.18784
  3    0    7   0  493.7707 0.587034 0.59297 0.06250  1.18977
  4    0    8   0  493.6340 0.136732 0.13945 0.00000  1.16785
  5    0    9   0  493.6161 0.017863 0.02264 0.00000  1.11581
  6    0   10   0  493.6154 7.5e-004  1e-003 0.00000  1.03602
  7    0   11   0  493.6154 2.4e-006  4e-006 0.00000  1.00229

Successful Termination After     7 Iterations
GCONV convergence criterion satisfied.
Criterion       493.6153781        Max Grad Entry  3.8624e-006
N. Active Constraints     0        Ridge (lambda)  0.000000000
Act.dF/Pred.dF  1.002288427
N. Function Calls        12        Preproces. Time           0
Time for Method           0        Effective Time            0

                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

      Parameter      Estimate   Gradient   Active BC

        1 v1_1     0.44691123 -9.83e-007
        2 v1_2    -0.29603115  9.26e-007
        3 v2_1     1.00293135 -3.78e-006
        4 v2_2    -1.01008998  3.86e-006
        5 v3_1     0.47298358 -4.04e-007
        6 v3_2    -0.28064655  2.76e-007
        7 v4_1     0.46313088 -1.84e-007
        8 v4_2    -0.09086462  3.47e-008
        9 v5_1     0.32886613 -1.88e-007
       10 v5_2    -0.71298315  2.29e-007
       11 Slope_1  1.50744512 -1.05e-006
       12 Slope_2  2.61692128 -3.70e-006
       13 Slope_3  0.87272121  7.76e-008
       14 Slope_4  0.46311043  1.51e-007
       15 Slope_5  0.67290523  1.88e-007

      Value of Objective Function =      493.615

           Negative Log-Likelihood = 493.615
```

```
          Estimates and Asymptotic Standard Errors
          ****************************************

                   Dense Matrix (15 by 2)

                      |    Estimate    AStdErr
                     -------------------------
                 v1_1 |   0.4469112  0.2918665
                 v1_2 |  -0.2960311  0.2854526
                 v2_1 |   1.0029314  0.5317309
                 v2_2 |  -1.0100900  0.5266266
                 v3_1 |   0.4729836  0.2387062
                 v3_2 |  -0.2806466  0.2343674
                 v4_1 |   0.4631309  0.2154338
                 v4_2 |  -0.0908646  0.2101593
                 v5_1 |   0.3288661  0.2226192
                 v5_2 |  -0.7129832  0.2333446
              Slope_1 |   1.5074451  0.5212142
              Slope_2 |   2.6169213  1.3433043
              Slope_3 |   0.8727212  0.3188570
              Slope_4 |   0.4631104  0.2618548
              Slope_5 |   0.6729052  0.2803141


        Statistic G2=987.231 df=227 Prob= 0.0000
        Statistic X2=250.024 df=227 Prob= 0.1407
         Statistic M2=40.071 df=35 Prob= 0.2553
```

Printing the return values:

```
print "GOF=", gof;
print "EST=", est;
print "COV=", cov;
```

```
           |          1
-----------------------
   Failure |   0.000000
   NegLike |   -493.615
   DetHess |    39.0907
        G2 |    987.231
      G2DF |    227.000
    G22PRB |   2.6e-095
      Chi2 |    250.024
    Chi2DF |    227.000
```

Chi2PRB	0.140722
M2	40.0710
M2DF	35.0000
M2PRB	0.255284
MAXMEM	23520.0
SUMMEM	43676.0
TotTime	0.000000
OptTime	0.000000
M2Time	0.000000
unused	0.000000
unused	0.000000
unused	0.000000

EST=

	Estimate	AStdErr
v1_1	0.44691	0.29187
v1_2	-0.29603	0.28545
v2_1	1.0029	0.53173
v2_2	-1.0101	0.52663
v3_1	0.47298	0.23871
v3_2	-0.28065	0.23437
v4_1	0.46313	0.21543
v4_2	-0.09086	0.21016
v5_1	0.32887	0.22262
v5_2	-0.71298	0.23334
Slope_1	1.5074	0.52121
Slope_2	2.6169	1.3433
Slope_3	0.87272	0.31886
Slope_4	0.46311	0.26185
Slope_5	0.67291	0.28031

COV=

SYM	v1_1	v1_2	v2_1	v2_2	v3_1
v1_1	0.08519				
v1_2	0.05983	0.08148			
v2_1	0.02837	0.04853	0.28274		
v2_2	0.05397	0.03437	-0.02411	0.27734	
v3_1	0.01394	0.01322	0.01778	0.02852	0.05698
v3_2	0.01340	0.01340	0.02428	0.02217	0.03939
v4_1	0.00718	0.00686	0.01006	0.01435	0.00406
v4_2	0.00722	0.00692	0.01127	0.01315	0.00412
v5_1	0.01033	0.01012	0.01597	0.02016	0.00605

```
    v5_2 |   0.01042   0.01009   0.02039   0.01482   0.00565
  Slope_1 |   0.03413  -0.01780  -0.08461   0.08323   0.00326
  Slope_2 |  -0.04655   0.02647   0.47454  -0.45637  -0.01960
  Slope_3 |   0.00283  -0.00079  -0.03039   0.03033   0.01158
  Slope_4 |  -0.00034  -0.00040  -0.01070   0.01076  -0.00053
  Slope_5 |  -0.00054   0.00017  -0.01629   0.01947   0.00159
             ..........................................


     SYM |   Slope_1   Slope_2   Slope_3   Slope_4   Slope_5
---------------------------------------------------------------
  Slope_1 |   0.27166
  Slope_2 |  -0.28098    1.8045
  Slope_3 |   0.02349  -0.09896   0.10167
  Slope_4 |   0.00310  -0.03403  -0.00019   0.06857
  Slope_5 |   0.00130  -0.05521   0.00913   0.00811   0.07858
```

2. Rasch model for SIM3 Data: N=67, n=5, m=3

```
  sim3fr = [
#include "..\\tdata\\sim3fr.dat"
          ];
  sim3fr = shape(sim3fr,.,6);
  cnam = [ "v1":"v5" "wgt" ];
  sim3fr = cname(sim3fr,cnam);

  bini = 1.030041;


  optn = [ "meth"    "rasch" ,
           "frequ"         6 ,
           "tech"   "nrridg" ,
           "phis"          3 ,
           "print"         3 ];
  < gof,est,cov > = irtml(sim3fr,optn,.,bini);
  print "GOF=", gof;
  print "EST=", est;
  print "COV=", cov;


          ********************************************
          IRT ML Estimation (Maydeu-Olivares & Joe, 2006)
                     Rasch Parametrized Model
          ********************************************

    Number Rows of Matrix . . . . . .          67
    Number Columns of Matrix  . . . . .         6
```

```
Number of Parameters. . . . . .          11
Total Number of Categories. . . . . .    15
Maximum Item Level. . . . . .             2
Column of Frequency Variable. . . . . .   6
Total Frequency . . . . . .             100
Number Uni- and Bivariate Moments . . . .  50
Number Integration Points . . . . . .      48
Integration Precision . . . . . .       1e-013
```

Initial Values for Thresholds

	1	2
v1	0.322773392	-0.200670695
v2	0.489548225	-0.532216814
v3	0.405465108	-0.241162057
v4	0.447312218	-0.080042708
v5	0.281851152	-0.663294217

Initial Values for Slope 1.03004

Newton-Raphson Ridge Optimization
Without Parameter Scaling
User Specified Gradient
User Specified Hessian (dense)

```
Iteration Start:
N. Variables             11
N. Bound. Constr.        22        N. Mask  Constr.          0
Criterion       500.1407956        Max Grad Entry  6.042285078
N. Active Constraints     0
```

Iter	rest	nfun	act	optcrit	difcrit	maxgrad	ridge	rho
1	0	2	0	498.1751	1.965682	0.84886	0.00000	1.09155
2	0	3	0	498.1219	0.053215	0.04659	0.00000	1.01137
3	0	4	0	498.1218	5.8e-005	2e-003	0.00000	0.98847
4	0	5	0	498.1218	4.3e-008	1e-004	0.00000	0.93581

```
Successful Termination After     4 Iterations
GCONV convergence criterion satisfied.
Criterion       498.1218409        Max Grad Entry  0.000118456
N. Active Constraints     0        Ridge (lambda)  0.000000000
Act.dF/Pred.dF  0.935807759
N. Function Calls         6        Preproces. Time           0
Time for Method           0        Effective Time            0
```

Negative Log-Likelihood = 498.122

Estimates and Asymptotic Standard Errors
****

Dense Matrix (11 by 2)

| Estimate | AStdErr
---|---|---
v1_1 | 0.3836869 | 0.2460784
v1_2 | -0.2647850 | 0.2447504
v2_1 | 0.6304398 | 0.2494394
v2_2 | -0.6484248 | 0.2511304
v3_1 | 0.4954470 | 0.2475083
v3_2 | -0.2875663 | 0.2450824
v4_1 | 0.5314206 | 0.2484822
v4_2 | -0.0931610 | 0.2435737
v5_1 | 0.3542736 | 0.2453474
v5_2 | -0.7916179 | 0.2545897
Slope | 1.0300443 | 0.1635387

Statistic G2=996.244 df=231 Prob= 0.0000
Statistic X2=259.007 df=231 Prob= 0.0995
Statistic M2=48.2485 df=39 Prob= 0.1472

GOF=

| 1
---|---
Failure | 0.000000
NegLike | -498.122
DetHess | 35.4053
G2 | 996.244
G2DF | 231.000
G22PRB | 1.6e-095
Chi2 | 259.007
Chi2DF | 231.000
Chi2PRB | 0.10
M2 | 48.2485
M2DF | 39.0000
M2PRB | 0.147215
MAXMEM | 23520.0
SUMMEM | 41916.0

```
 TotTime |   0.000000
 OptTime |   0.000000
  M2Time |   0.000000
  unused |   0.000000
  unused |   0.000000
  unused |   0.000000
```

3. Samejima model for LSAT6 Data: Compute Godambe Matrix, standardized marginal residuals and the Σ_r matrix:

```
optn = [ "meth"     "same" ,
         "cl"       "goda" ,
         "vardef"      "N" ,
         "tech"   "nrridg" ,
         "phis"          2 ,
         "pcov"            ,
         "pres"            ,
         "print"         3 ];
< gof,est,cov,resi,stmres,sigmar > = irtml(lsat6,optn);
```

```
[note] Data set was reduced from 1000 rows to 30 distinctive rows
       and a frequency column was added.
Data set was reduced from 1000 rows to 30 distinctive rows and a
                 frequency column was added.


        ********************************************
        IRT ML Estimation (Maydeu-Olivares & Joe, 2006)
               Samejima Graded Model: LOGIT Link
        ********************************************

  Number Rows of Matrix . . . . . .          30
  Number Columns of Matrix  . . . . .         6
  Number of Parameters. . . . . .            10
  Total Number of Categories. . . . . .      10
  Maximum Item Level. . . . . .               1
  Column of Frequency Variable. . . . . .     6
  Total Frequency . . . . . .              1000
  Number Uni- and Bivariate Moments . . . .   50
  Number Uni- and Bivariate Moments (Red) . .  15
  Number Integration Points . . . . . .       48


                 ******************************
                 Frequencies of Data Categories
                 ******************************
```

Variable	Level	Frequency	Percentage
i1	0	76	7.60 %
	1	924	92.40 %
i2	0	291	29.10 %
	1	709	70.90 %
i3	0	447	44.70 %
	1	553	55.30 %
i4	0	237	23.70 %
	1	763	76.30 %
i5	0	130	13.00 %
	1	870	87.00 %

Initial Values for Thresholds

	1
i1	2.497978731
i2	0.890532259
i3	0.212799407
i4	1.169197890
i5	1.900958761

Initial Values for Slopes

	1
i1	1.000000000
i2	1.000000000
i3	1.000000000
i4	1.000000000
i5	1.000000000

Newton-Raphson Ridge Optimization
Without Parameter Scaling
User Specified Gradient
User Specified Hessian (dense)

Iteration Start:			
N. Variables	10		
N. Bound. Constr.	15	N. Mask Constr.	0
Criterion	2484.340596	Max Grad Entry	25.22782234
N. Active Constraints	0		

Iter	rest	nfun	act	optcrit	difcrit	maxgrad	ridge	rho
1	0	2	0	2468.415	15.92549	10.2273	0.00000	0.76061
2	0	3	0	2466.687	1.727711	0.78210	0.00000	0.95275
3	0	4	0	2466.654	0.033836	0.07628	0.00000	1.03962
4	0	5	0	2466.653	1.8e-004	6e-004	0.00000	1.00483
5	0	6	0	2466.653	1.0e-008	3e-008	0.00000	1.00001

Successful Termination After	5 Iterations		
GCONV convergence criterion satisfied.			
Criterion	2466.653377	Max Grad Entry	3.2229e-008
N. Active Constraints	0	Ridge (lambda)	0.000000000
Act.dF/Pred.dF	1.000012492		
N. Function Calls	7	N. Gradient Calls	5
N. Hessian Calls	7	Preproces. Time	0
Time for Method	0	Effective Time	0

Godambe Matrix

	i1_1	i2_1	i3_1	i4_1	i5_1
i1_1	1.4054858	0.0054481	0.0141442	0.0035181	-5.48e-004
i2_1	0.0054481	0.2684306	0.0147390	0.0094219	0.0083117
i3_1	0.0141442	0.0147390	0.1934758	0.0140000	0.0133094
i4_1	0.0035181	0.0094219	0.0140000	0.3228195	0.0073253
i5_1	-5.48e-004	0.0083117	0.0133094	0.0073253	0.5861221
Slope_1	1.3916070	-0.0223329	-0.0158560	-0.0238627	-0.0287817
Slope_2	-0.0582286	0.2868121	-0.0268028	-0.0285728	-0.0301332
Slope_3	-0.1548064	-0.0965308	0.1234925	-0.0928640	-0.0906613
Slope_4	-0.0498931	-0.0229937	-0.0206717	0.3423425	-0.0249792
Slope_5	-0.0416575	-0.0167988	-0.0137887	-0.0173249	0.5914877

Godambe Matrix

	Slope_1	Slope_2	Slope_3	Slope_4	Slope_5
i1_1	1.3916070	-0.0582286	-0.1548064	-0.0498931	-0.0416575
i2_1	-0.0223329	0.2868121	-0.0965308	-0.0229937	-0.0167988
i3_1	-0.0158560	-0.0268028	0.1234925	-0.0206717	-0.0137887
i4_1	-0.0238627	-0.0285728	-0.0928640	0.3423425	-0.0173249
i5_1	-0.0287817	-0.0301332	-0.0906613	-0.0249792	0.5914877
Slope_1	2.2080493	-0.0807754	-0.2315846	-0.0679387	-0.0550071
Slope_2	-0.0807754	1.1349243	-0.3680511	-0.0830729	-0.0596606
Slope_3	-0.2315846	-0.3680511	1.7175640	-0.2855510	-0.1946239
Slope_4	-0.0679387	-0.0830729	-0.2855510	1.0995847	-0.0483682
Slope_5	-0.0550071	-0.0596606	-0.1946239	-0.0483682	1.3428420

Fisher Scoring Matrix

	i1_1	i2_1	i3_1	i4_1	i5_1
i1_1	1.4038558	0.0053614	0.0141805	0.0033227	-9.89e-004
i2_1	0.0053614	0.2683434	0.0147363	0.0094420	0.0082923
i3_1	0.0141805	0.0147363	0.1934724	0.0140041	0.0133268
i4_1	0.0033227	0.0094420	0.0140041	0.3226690	0.0072510
i5_1	-9.89e-004	0.0082923	0.0133268	0.0072510	0.5857445
Slope_1	1.3889664	-0.0224822	-0.0157984	-0.0241882	-0.0295066
Slope_2	-0.0584678	0.2864821	-0.0268149	-0.0284711	-0.0301691
Slope_3	-0.1535884	-0.0964608	0.1234384	-0.0926504	-0.0901405
Slope_4	-0.0504661	-0.0229224	-0.0206597	0.3418718	-0.0251963
Slope_5	-0.0426717	-0.0168463	-0.0137495	-0.0174974	0.5906284

Fisher Scoring Matrix

	Slope_1	Slope_2	Slope_3	Slope_4	Slope_5
i1_1	1.3889664	-0.0584678	-0.1535884	-0.0504661	-0.0426717
i2_1	-0.0224822	0.2864821	-0.0964608	-0.0229224	-0.0168463
i3_1	-0.0157984	-0.0268149	0.1234384	-0.0206597	-0.0137495
i4_1	-0.0241882	-0.0284711	-0.0926504	0.3418718	-0.0174974
i5_1	-0.0295066	-0.0301691	-0.0901405	-0.0251963	0.5906284
Slope_1	2.2037684	-0.0811954	-0.2296060	-0.0688944	-0.0566735
Slope_2	-0.0811954	1.1336679	-0.3678633	-0.0827252	-0.0597543
Slope_3	-0.2296060	-0.3678633	1.7162309	-0.2849138	-0.1934351
Slope_4	-0.0688944	-0.0827252	-0.2849138	1.0981108	-0.0488724
Slope_5	-0.0566735	-0.0597543	-0.1934351	-0.0488724	1.3408860

Ratio of Diagonals: Fisher Scoring / Godambe Matrix

1 :	0.9988	0.9997	1	0.9995	0.9994
6 :	0.9981	0.9989	0.9992	0.9987	0.9985

Log-Likelihood = -2466.65
Statistic G2=4933.31 df=21 Prob= 0.0000
Statistic X2=18.141 df=21 Prob= 0.6401
Statistic M2=4.73838 df=5 Prob= 0.4486

Estimates and Asymptotic Standard Errors
****

Dense Matrix (10 by 7)

	Estimate	Wald_ASE	Wald_LCI	Wald_UCI	Goda_ASE
i1_1	2.7732344	0.2057437	2.3699841	3.1764847	1.1855319
i2_1	0.9902012	0.0900190	0.8137673	1.1666351	0.5181029
i3_1	0.2491475	0.0762729	0.0996554	0.3986396	0.4398589
i4_1	1.2847569	0.0990375	1.0906470	1.4788669	0.5681720
i5_1	2.0532704	0.1353588	1.7879721	2.3185688	0.7655861
Slope_1	0.8256595	0.2581149	0.3197635	1.3315555	1.4859506
Slope_2	0.7227442	0.1866797	0.3568588	1.0886296	1.0653282
Slope_3	0.8908745	0.2327638	0.4346658	1.3470833	1.3105587
Slope_4	0.6883680	0.1851429	0.3254945	1.0512415	1.0486108
Slope_5	0.6568559	0.2099092	0.2454415	1.0682704	1.1588106

	Goda_LCI	Goda_UCI
i1_1	0.4496347	5.0968341
i2_1	-0.0252619	2.0056643
i3_1	-0.6129600	1.1112551
i4_1	0.1711602	2.3983537
i5_1	0.5527493	3.5537916
Slope_1	-2.0867502	3.7380692
Slope_2	-1.3652608	2.8107492
Slope_3	-1.6777733	3.4595223
Slope_4	-1.3668715	2.7436075
Slope_5	-1.6143711	2.9280830

Inverse Hessian Matrix

	i1_1	i2_1	i3_1	i4_1	i5_1
i1_1	0.0423305	2.01e-004	7.94e-004	-8.47e-004	-0.0017665
i2_1	2.01e-004	0.0081034	4.33e-004	3.63e-005	6.82e-004
i3_1	7.94e-004	4.33e-004	0.0058176	4.24e-004	9.93e-005
i4_1	-8.47e-004	3.63e-005	4.24e-004	0.0098084	0.0010823
i5_1	-0.0017665	6.82e-004	9.93e-005	0.0010823	0.0183220
Slope_1	0.0419994	-6.10e-004	1.06e-004	-0.0022183	-0.0036214
Slope_2	-0.0015874	0.0088056	-8.41e-004	-0.0018246	7.96e-004
Slope_3	5.48e-004	-0.0030363	0.0038944	-0.0027246	-0.0068738
Slope_4	-0.0045305	-0.0014765	-6.07e-004	0.0106699	0.0020030
Slope_5	-0.0052033	4.73e-004	-0.0010911	0.0014274	0.0194141

Inverse Hessian Matrix

	Slope_1	Slope_2	Slope_3	Slope_4	Slope_5
i1_1	0.0419994	-0.0015874	5.48e-004	-0.0045305	-0.0052033

i2_1	-6.10e-004	0.0088056	-0.0030363	-0.0014765	4.73e-004
i3_1	1.06e-004	-8.41e-004	0.0038944	-6.07e-004	-0.0010911
i4_1	-0.0022183	-0.0018246	-0.0027246	0.0106699	0.0014274
i5_1	-0.0036214	7.96e-004	-0.0068738	0.0020030	0.0194141
Slope_1	0.0666233	-0.0021663	0.0012366	-0.0068227	-0.0078807
Slope_2	-0.0021663	0.0348493	-0.0115996	-0.0055814	0.0020499
Slope_3	0.0012366	-0.0115996	0.0541790	-0.0083753	-0.0152185
Slope_4	-0.0068227	-0.0055814	-0.0083753	0.0342779	0.0047624
Slope_5	-0.0078807	0.0020499	-0.0152185	0.0047624	0.0440619

Observed and Predicted Values and Residuals

****

Dense Matrix (30 by 3)

	Predict	Observed	Residual
1	0.0022765	0.0030000	7.24e-004
2	0.0058622	0.0060000	1.38e-004
3	0.0025957	0.0020000	-5.96e-004
4	0.0089446	0.0110000	0.0020554
5	6.96e-004	0.0010000	3.04e-004
6	0.0026126	0.0010000	-0.0016126
7	0.0011781	0.0030000	0.0018219
8	0.0059531	0.0040000	-0.0019531
9	0.0018398	0.0010000	-8.40e-004
10	0.0064333	0.0080000	0.0015667
11	0.0135796	0.0160000	0.0024204
12	0.0043693	0.0030000	-0.0013693
13	0.0020002	0.0020000	-2.15e-007
14	0.0139159	0.0150000	0.0010841
15	0.0094755	0.0100000	5.24e-004
16	0.0346169	0.0290000	-0.0056169
17	0.0155886	0.0140000	-0.0015886
18	0.0765680	0.0810000	0.0044320
19	0.0046563	0.0030000	-0.0016563
20	0.0249833	0.0280000	0.0030167
21	0.0114622	0.0150000	0.0035378
22	0.0835365	0.0800000	-0.0035365
23	0.0112545	0.0160000	0.0047455
24	0.0561126	0.0560000	-1.13e-004
25	0.0256523	0.0210000	-0.0046523
26	0.1733138	0.1730000	-3.14e-004
27	0.0084457	0.0110000	0.0025543
28	0.0625185	0.0610000	-0.0015185

29	0.0291373	0.0280000 -0.0011373
30	0.2966789	0.2980000 0.0013211

Standardized Marginal Residuals (First 5: Univariate Moments)

Dense Matrix (15 by 4)

	Predictd	Observed	ASTResid	AsStdErr
1	0.9240009	0.9240000	0.0461629	1.88e-005
2	0.7089944	0.7090000	-0.2107603	2.64e-005
3	0.5529952	0.5530000	-0.0774121	6.18e-005
4	0.7629957	0.7630000	-0.3003791	1.44e-005
5	0.8699991	0.8700000	-0.1521033	6.05e-006
6	0.6631136	0.6640000	-0.2910838	0.0030450
7	0.5214187	0.5240000	-0.9805793	0.0026324
8	0.4179176	0.4180000	-0.0247710	0.0033259
9	0.7119376	0.7100000	0.6325768	0.0030631
10	0.5571689	0.5530000	1.0164146	0.0041016
11	0.4438622	0.4450000	-0.3292587	0.0034556
12	0.8083285	0.8060000	0.8477122	0.0027468
13	0.6269219	0.6300000	-0.8092381	0.0038037
14	0.4945681	0.4900000	1.3758320	0.0033203
15	0.6724904	0.6780000	-1.4460266	0.0038102

Matrix Sigma_r (First 5: Univariate Moments)

Dense Symmetric Matrix (15 by 15)

S	1	2	3	4	5
1	3.17e-007				
2	-4.35e-007	6.26e-007			
3	-1.04e-006	1.44e-006	3.44e-006		
4	-2.29e-007	3.39e-007	7.65e-007	1.86e-007	
5	9.88e-008	-1.28e-007	-3.22e-007	-6.49e-008	3.30e-008
6	-1.99e-006	2.63e-006	6.48e-006	1.35e-006	-6.50e-007
7	-1.21e-006	-2.85e-009	3.26e-006	-5.40e-007	-8.31e-007
8	-2.00e-006	4.45e-006	7.33e-006	2.90e-006	-1.59e-007
9	-2.19e-006	3.81e-006	7.54e-006	2.27e-006	-4.65e-007
10	-1.85e-006	1.02e-006	5.48e-006	4.12e-008	-9.98e-007
11	-1.59e-006	2.72e-006	5.47e-006	1.61e-006	-3.48e-007
12	-1.81e-006	4.55e-006	6.90e-006	3.06e-006	-4.13e-009

```
    13 |  -1.90e-006  1.81e-006  5.89e-006  6.99e-007 -8.09e-007
    14 |  -9.62e-007  3.52e-007  2.73e-006 -1.28e-007 -5.64e-007
    15 |  -2.24e-006  2.97e-006  7.30e-006  1.53e-006 -7.25e-007

     S |           6           7           8           9          10
     ------------------------------------------------------------
     6 |   0.0083450
     7 |  -0.0031167  0.0062367
     8 |  -0.0033041 -0.0026858  0.0099557
     9 |  -0.0028402 -0.0028971  0.0045266  0.0084441
    10 |  -0.0034494  0.0046887 -0.0051858 -0.0035053  0.0151409

     S |           6           7           8           9          10
     ------------------------------------------------------------
    11 |   0.0046241 -0.0026919 -0.0040043 -0.0033880 -0.0057476
    12 |  -0.0019139 -0.0020534  0.0026466 -0.0016946  0.0010964
    13 |  -0.0021987  0.0026914 -0.0040406  0.0010750 -0.0044310
    14 |   0.0027268 -0.0019417 -0.0034297  0.0022061  0.0059843
    15 |   0.0010683  0.0021639  0.0056424 -0.0019484 -0.0044220

     S |          11          12          13          14          15
     ------------------------------------------------------------
    11 |   0.0107470
    12 |   0.0021870  0.0067905
    13 |   0.0058133 -0.0023566  0.0130217
    14 |  -0.0034829 -0.0026569 -0.0049607  0.0099217
    15 |  -0.0041242 -0.0020536 -0.0043749 -0.0045670  0.0130656

  print "GOF=", gof;

GOF=
            |          1
------------------------
   Failure  |   0.000000
   NegLike  |   -2466.65
   DetHess  |   -39.9939
        G2  |    4933.31
      G2DF  |    21.0000
    G22PRB  |   0.000000
      Chi2  |    18.1410
     Chi2DF |    21.0000
   Chi2PRB  |   0.640067
        M2  |    4.73838
```

```
    M2DF |    5.00000
    M2PRB |   0.448635
   MAXMEM |    17640.0
   SUMMEM |    27052.0
  TotTime |   0.000000
  OptTime |   0.000000
   M2Time |   0.000000
   unused |   0.000000
   unused |   0.000000
   unused |   0.000000
```

4. Samejima parametrized model for BHS Data: N=239, n=20, m=2

```
  bhsfr = [
#include "..\\tdata\\bhsfr.dat"
          ];
  bhsfr = shape(bhsfr,.,21);
  cnam = [ "v1":"v20" "wgt" ];
  bhsfr = cname(bhsfr,cnam);

  tini = [ -3.621505  0.206505 -3.343179 -1.963415 -5.527023
           -2.287424 -1.983795 -4.305788 -4.573405 -0.536651
           -6.196920 -2.819038 -2.869461 -0.541555 -4.105525
            0.012029 -1.691366 -1.524191 -1.662112 -3.198459 ];
  bini = [  2.304435  0.936715  2.623722  1.567783  3.788161
            2.359669  2.828300  2.748323  2.638820  1.609857
            4.176470  1.958716  1.149285  0.965500  2.118243
            0.776834  1.134731  0.322436  1.497783  2.096906 ];
```

The starting values for thresholds are computed automatically:

```
  /* NRRIDG: niter=7, nfun=9 */
  optn = [ "meth"     "same" ,
           "frequ"        21 ,
           "tech"   "nrridg" ,
           "phis"          3 ,
           "print"         3 ];
  < gof,est,cov > = irtml(bhsfr,optn,.,bini);
  print "GOF=", gof;
  print "EST=", est;
  print "COV=", cov;
```

```
          ********************************************
          IRT ML Estimation (Maydeu-Olivares & Joe, 2006)
                      Fully Parametrized Model
```

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

Number Rows of Matrix . . . . . .          239
Number Columns of Matrix  . . . . . .       21
Number of Parameters. . . . . .             40
Total Number of Categories. . . . . .       40
Maximum Item Level. . . . . .                1
Column of Frequency Variable. . . . . .     21
Total Frequency . . . . . .                393
Number Uni- and Bivariate Moments . . . . .  210
Number Integration Points . . . . . .        48
Integration Precision . . . . . .        1e-013
```

Frequencies of Data Categories

Variable	Level	Frequency	Percentage
v1	0	353	89.82 %
	1	40	10.18 %
v2	0	180	45.80 %
	1	213	54.20 %
v3	0	338	86.01 %
	1	55	13.99 %
v4	0	316	80.41 %
	1	77	19.59 %
v5	0	359	91.35 %
	1	34	8.65 %
v6	0	309	78.63 %
	1	84	21.37 %
v7	0	286	72.77 %
	1	107	27.23 %
v8	0	358	91.09 %
	1	35	8.91 %
v9	0	365	92.88 %
	1	28	7.12 %
v10	0	233	59.29 %
	1	160	40.71 %
v11	0	362	92.11 %
	1	31	7.89 %
v12	0	339	86.26 %
	1	54	13.74 %
v13	0	360	91.60 %
	1	33	8.40 %

v14	0	241	61.32 %
	1	152	38.68 %
v15	0	367	93.38 %
	1	26	6.62 %
v16	0	196	49.87 %
	1	197	50.13 %
v17	0	314	79.90 %
	1	79	20.10 %
v18	0	321	81.68 %
	1	72	18.32 %
v19	0	303	77.10 %
	1	90	22.90 %
v20	0	347	88.30 %
	1	46	11.70 %

Initial Values for Thresholds

	1
v1	-2.177588603
v2	0.168335315
v3	-1.815712710
v4	-1.411936792
v5	-2.356961864
v6	-1.302524478
v7	-0.983162976
v8	-2.325184925
v9	-2.567692843
v10	-0.375864638
v11	-2.457657007
v12	-1.837016061
v13	-2.389596470
v14	-0.460916413
v15	-2.647265310
v16	0.005089070
v17	-1.379945133
v18	-1.494775004
v19	-1.213923135
v20	-2.020683383

Initial Values for Slopes

	1
v1	2.304435000
v2	0.936715000
v3	2.623722000
v4	1.567783000
v5	3.788161000
v6	2.359669000
v7	2.828300000
v8	2.748323000
v9	2.638820000
v10	1.609857000
v11	4.176470000
v12	1.958716000
v13	1.149285000
v14	0.965500000
v15	2.118243000
v16	0.776834000
v17	1.134731000
v18	0.322436000
v19	1.497783000
v20	2.096906000

Newton-Raphson Ridge Optimization
Without Parameter Scaling
User Specified Gradient
User Specified Hessian (dense)

Iteration Start:
N. Variables 40

```
 N. Bound. Constr.        60        N. Mask  Constr.          0
 Criterion       3036.375933        Max Grad Entry  30.65645603
 N. Active Constraints     0


Iter rest nfun act   optcrit  difcrit maxgrad   ridge      rho
   1    0    3   0  2949.307 87.06909 4.91791 8.00000  1.23937
   2    0    4   0  2932.595 16.71178 2.19171 2.00000  1.26536
   3    0    5   0  2928.965 3.630345 0.83451 0.50000  1.28838
   4    0    6   0  2928.155 0.809451 0.21924 0.12500  1.19485
   5    0    7   0  2928.105 0.050473 0.05047 0.03125  0.63552
   6    0    8   0  2928.102 3.2e-003  2e-003 0.00000  0.81445
   7    0    9   0  2928.102 5.7e-006  6e-006 0.00000  1.00102

 Successful Termination After     7 Iterations
 GCONV convergence criterion satisfied.
 Criterion       2928.101577        Max Grad Entry  6.2969e-006
 N. Active Constraints     0        Ridge (lambda)  0.000000000
 Act.dF/Pred.dF  1.001021344
 N. Function Calls        10        Preproces. Time           3
 Time for Method           5        Effective Time            9

            Negative Log-Likelihood = 2928.1

          Estimates and Asymptotic Standard Errors
          ****************************************

                  Dense Matrix (40 by 2)

                      |    Estimate    AStdErr
                    ---------------------------
                 v1_1 |  -3.6220554  0.4362764
                 v2_1 |   0.1980487  0.1202594
                 v3_1 |  -3.3354008  0.4205838
                 v4_1 |  -1.9677746  0.2135887
                 v5_1 |  -5.5071734  0.9261131
                 v6_1 |  -2.3134939  0.2982581
                 v7_1 |  -2.0337675  0.3187383
                 v8_1 |  -4.3188424  0.5964614
                 v9_1 |  -4.5778373  0.6216664
                v10_1 |  -0.5395725  0.1556139
                v11_1 |  -6.1971788  1.1637369
                v12_1 |  -2.8296063  0.3165626
                v13_1 |  -2.8688770  0.2714759
                v14_1 |  -0.5415008  0.1264814
```

```
                    v15_1 |  -4.1338423  0.5072403
                    v16_1 |   0.0160516  0.1152506
                    v17_1 |  -1.6995983  0.1749185
                    v18_1 |  -1.5231760  0.1358046
                    v19_1 |  -1.6709722  0.1943453
                    v20_1 |  -3.1984990  0.3648423
                 Slope_01 |   2.2545474  0.3985612
                 Slope_02 |   0.9246440  0.1588923
                 Slope_03 |   2.5696999  0.4243659
                 Slope_04 |   1.5377179  0.2423900
                 Slope_05 |   3.7000000  0.7994154
                 Slope_06 |   2.4018419  0.3607215
                 Slope_07 |   2.9325881  0.4568444
                 Slope_08 |   2.7504465  0.5331429
                 Slope_09 |   2.6310134  0.5138176
                 Slope_10 |   1.6472600  0.2345896
                 Slope_11 |   4.0957225  0.9719788
                 Slope_12 |   1.9681089  0.3199236
                 Slope_13 |   1.1452687  0.2470946
                 Slope_14 |   0.9818490  0.1642769
                 Slope_15 |   2.0820172  0.3997149
                 Slope_16 |   0.7946752  0.1493851
                 Slope_17 |   1.1130764  0.1941790
                 Slope_18 |   0.3282404  0.1517152
                 Slope_19 |   1.5165112  0.2305903
                 Slope_20 |   2.0510978  0.3472774


          Statistic G2=5856.2 df=1048535 Prob= 1.0000
       Statistic X2=3.3495e+006 df=1048535 Prob= 0.0000
          Statistic M2=232.108 df=170 Prob= 0.0011

GOF=
           |           1
-----------------------
   Failure |   0.0000000
   NegLike |   -2928.102
   DetHess |    115.5289
        G2 |    5856.203
      G2DF |   1048535.0
    G22PRB |   1.0000000
      Chi2 |   3349499.7
     Chi2DF |   1048535.0
   Chi2PRB |   0.0000000
        M2 |    232.1083
      M2DF |    170.0000
```

```
  M2PRB |   1.09e-003
 MAXMEM |    403584.0
 SUMMEM |    670288.0
TotTime |    11.00000
OptTime |    8.000000
 M2Time |    3.000000
 unused |   0.0000000
 unused |   0.0000000
 unused |   0.0000000
```

5. Rasch model for BHS Data: N=239, n=20, m=2

```
tini = [ -2.905566  0.261941 -2.435364 -1.900024 -3.135155
         -1.753147 -1.320564 -3.094652 -3.401888 -0.486926
         -3.263016 -2.463301 -3.176675 -0.604159 -3.501757
          0.037908 -1.857154 -2.010733 -1.633686 -2.702835 ];
bini = 1.465312;


optn = [ "meth"    "rasch" ,
         "frequ"        21 ,
         "tech"   "nrridg" ,
         "phis"          3 ,
         "print"         3 ];
< gof,est,cov > = irtml(bhsfr,optn,tini,bini);
print "GOF=", gof;
print "EST=", est;
print "COV=", cov;


        ********************************************
        IRT ML Estimation (Maydeu-Olivares & Joe, 2006)
                    Rasch Parametrized Model
        ********************************************

  Number Rows of Matrix . . . . . .        239
  Number Columns of Matrix  . . . . .       21
  Number of Parameters. . . . . .           21
  Total Number of Categories. . . . . .     40
  Maximum Item Level. . . . . .              1
  Column of Frequency Variable. . . . . .   21
  Total Frequency . . . . . .              393
  Number Uni- and Bivariate Moments . . . .  210
  Number Integration Points . . . . . .      48
  Integration Precision . . . . . .      1e-013
```

Initial Values for Thresholds

	1
v1	-2.177588603
v2	0.168335315
v3	-1.815712710
v4	-1.411936792
v5	-2.356961864
v6	-1.302524478
v7	-0.983162976
v8	-2.325184925
v9	-2.567692843
v10	-0.375864638
v11	-2.457657007
v12	-1.837016061
v13	-2.389596470
v14	-0.460916413
v15	-2.647265310
v16	0.005089070
v17	-1.379945133
v18	-1.494775004
v19	-1.213923135
v20	-2.020683383

Initial Values for Slope 1.46531

Newton-Raphson Ridge Optimization
Without Parameter Scaling
User Specified Gradient
User Specified Hessian (dense)

Iteration Start:			
N. Variables	21		
N. Bound. Constr.	42	N. Mask Constr.	0
Criterion	3066.714934	Max Grad Entry	27.88228402
N. Active Constraints	0		

Iter	rest	nfun	act	optcrit	difcrit	maxgrad	ridge	rho
1	0	2	0	3026.796	39.91886	6.92858	0.00000	0.94616
2	0	3	0	3025.508	1.287732	6.40640	0.00000	1.09909
3	0	4	0	3025.425	0.083552	3.51083	0.00000	1.51285
4	0	5	0	3025.413	0.012055	1.89498	0.00000	0.77275
5	0	6	0	3025.406	6.8e-003	1.02700	0.00000	1.46302
6	0	7	0	3025.404	2.1e-003	0.55432	0.00000	1.51820

```
  7    0    8    0  3025.403 5.9e-004 0.29957 0.00000  1.49793
  8    0    9    0  3025.403 1.6e-004 0.16223 0.00000  1.34794
  9    0   19    0  3025.403 1.0e-005 0.14911 2048.00  1.91922

Successful Termination After     9 Iterations
GCONV convergence criterion satisfied.
Criterion       3025.403101        Max Grad Entry  0.149105707
N. Active Constraints     0        Ridge (lambda)  512.0000000
Act.dF/Pred.dF  1.919216227
N. Function Calls        20        Preproces. Time           4
Time for Method          11        Effective Time           16
```

Negative Log-Likelihood = 3025.4

Estimates and Asymptotic Standard Errors

****

Dense Matrix (21 by 2)

	Estimate	AStdErr
v1_1	-2.9493744	0.2085342
v2_1	0.2600489	0.1403274
v3_1	-2.4678718	0.1862272
v4_1	-1.9229986	0.1674480
v5_1	-3.2287026	0.2244092
v6_1	-1.7739727	0.1633307
v7_1	-1.3359544	0.1534272
v8_1	-3.1440379	0.2193480
v9_1	-3.4619607	0.2395584
v10_1	-0.4944606	0.1423733
v11_1	-3.3645462	0.2330100
v12_1	-2.4678718	0.1862272
v13_1	-3.2287026	0.2244092
v14_1	-0.5977657	0.1432169
v15_1	-3.5658124	0.2469066
v16_1	0.0484731	0.1401716
v17_1	-1.8794830	0.1662039
v18_1	-2.0125144	0.1701191
v19_1	-1.6528860	0.1602754
v20_1	-2.7413659	0.1981642
Slope	1.4750370	0.0509284

Statistic G2=6050.81 df=1048554 Prob= 1.0000

```
        Statistic X2=1.27914e+007 df=1048554 Prob= 0.0000
             Statistic M2=444.12 df=189 Prob= 0.0000


           |           1
-------------------------
   Failure |   0.0000000
   NegLike |   -3025.403
   DetHess |    76.57134
        G2 |    6050.806
      G2DF |     1048554
     G22PRB |   1.0000000
       Chi2 |    12791371
     Chi2DF |     1048554
    Chi2PRB |   0.0000000
        M2 |    444.1203
      M2DF |    189.0000
     M2PRB |   1.38e-022
     MAXMEM |    403584.0
     SUMMEM |    637608.0
    TotTime |    18.00000
    OptTime |    16.00000
     M2Time |    2.000000
     unused |   0.0000000
     unused |   0.0000000
     unused |   0.0000000
```

6. Running the EM algorithm: SIM3 Data: N=67, n=5, m=3:

```
  print "SIM3FR: nobs=67, nvar=5, m=3 cats and Frequency: EM";
  options NOECHO;
  sim3fr = [
#include "..\\tdata\\sim3fr.dat"
           ];
  options ECHO;
  sim3fr = shape(sim3fr,.,6);
  cnam = [ "v1":"v5" "wgt" ];
  sim3fr = cname(sim3fr,cnam);


  optn = [ "meth"       "em" ,
           "frequ"         6 ,
           "tech"   "nrridg" ,
           "phis"          2 ,
           "print"         3 ];
  < gof,est,cov > = irtml(sim3fr,optn);
```

```
print "GOF=", gof;
print "EST=", est;
print "COV=", cov;
```

```
  ********************************************************
  EM Algorithm of IRT ML Estimation (Bock and Aitkin, 1981)
            Samejima Graded Model: LOGIT Link
  ********************************************************

 Number Rows of Matrix . . . . . .          67
 Number Columns of Matrix  . . . . .         6
 Number of Parameters. . . . . .             15
 Total Number of Categories. . . . . .       15
 Maximum Item Level. . . . . .                2
 Column of Frequency Variable. . . . . .      6
 Total Frequency . . . . . .                100
 Number Uni- and Bivariate Moments . . . . .  50
 Number Integration Points . . . . . .        48
```

Frequencies of Data Categories

Variable	Level	Frequency	Percentage
v1	0	42	42.00 %
	1	13	13.00 %
	2	45	45.00 %
v2	0	38	38.00 %
	1	25	25.00 %
	2	37	37.00 %
v3	0	40	40.00 %
	1	16	16.00 %
	2	44	44.00 %
v4	0	39	39.00 %
	1	13	13.00 %
	2	48	48.00 %
v5	0	43	43.00 %
	1	23	23.00 %
	2	34	34.00 %

Initial Values for Thresholds

	1	2
v1	0.322773392	-0.200670695

```
v2   0.489548225 -0.532216814
v3   0.405465108 -0.241162057
v4   0.447312218 -0.080042708
v5   0.281851152 -0.663294217
```

Initial Values for Slopes

```
                    1
v1   1.000000000
v2   1.000000000
v3   1.000000000
v4   1.000000000
v5   1.000000000
```

History of Bock-Aitkin EM Algorithm

Iter	ItrMst	Likelihood	Dif_Tresh	Dif_Slope
1	45	93.4399055	0.16934758	0.25201326
2	41	89.7185598	0.09505603	0.20877853
3	42	88.6013669	0.08132042	0.17456148
4	48	87.9387388	0.06331168	0.14555744
5	46	87.5528689	0.04507469	0.12027428

Iter	ItrMst	Likelihood	Dif_Tresh	Dif_Slope
101	1	87.0068421	1.311e-005	1.284e-004
102	1	87.0068412	1.270e-005	1.268e-004
103	1	87.0068404	1.230e-005	1.252e-004
104	1	87.0068396	1.192e-005	1.237e-004
105	1	87.0068388	1.155e-005	1.221e-004
106	1	87.0068380	1.119e-005	1.206e-004
107	1	87.0068373	1.085e-005	1.191e-004
108	1	87.0068365	1.052e-005	1.176e-004
109	1	87.0068358	1.021e-005	1.161e-004
110	1	87.0068352	9.902e-006	1.146e-004
111	1	87.0068345	9.610e-006	1.132e-004
112	1	87.0068338	9.328e-006	1.118e-004
113	1	87.0068332	9.057e-006	1.104e-004
114	1	87.0068326	8.795e-006	1.090e-004
115	1	87.0068320	8.544e-006	1.076e-004
116	1	87.0068314	8.301e-006	1.062e-004
117	1	87.0068309	8.067e-006	1.048e-004
118	1	87.0068303	7.842e-006	1.035e-004
119	1	87.0068298	7.625e-006	1.022e-004
120	1	87.0068293	7.415e-006	1.009e-004

```
                121       1  87.0068288  7.213e-006  9.958e-005

                       Log-Likelihood = -493.615


                 Estimates and Asymptotic Standard Errors
                 ****************************************

                          Dense Matrix (15 by 2)

                             |    Estimate    AStdErr
                          ---------------------------
                          v1_1 |   0.4473400  0.2920436
                          v1_2 |  -0.2961067  0.2856577
                          v2_1 |   1.0011738  0.5291365
                          v2_2 |  -1.0076527  0.5234176
                          v3_1 |   0.4731773  0.2387282
                          v3_2 |  -0.2805420  0.2343853
                          v4_1 |   0.4632150  0.2154408
                          v4_2 |  -0.0907960  0.2101636
                          v5_1 |   0.3290131  0.2226284
                          v5_2 |  -0.7128725  0.2333414
                       Slope_1 |   1.5107359  0.5220222
                       Slope_2 |   2.6104809  1.3318188
                       Slope_3 |   0.8741315  0.3190489
                       Slope_4 |   0.4640826  0.2621965
                       Slope_5 |   0.6740072  0.2806052


                  Statistic G2=987.229 df=227 Prob= 0.0000
                   Statistic X2=250.01 df=227 Prob= 0.1409
                   Statistic M2=40.0763 df=35 Prob= 0.2551

GOF=
              |          1
   -----------------------
      Failure |   0.000000
      NegLike |   -493.615
      DetHess |    39.1003
           G2 |    987.229
         G2DF |    227.000
       G22PRB |   2.6e-095
         Chi2 |    250.010
       Chi2DF |    227.000
      Chi2PRB |   0.140853
```

```
        M2 |    40.0763
      M2DF |    35.0000
     M2PRB |   0.255100
    MAXMEM |    23520.0
    SUMMEM |    43572.0
   TotTime |   1.000000
   OptTime |   1.000000
    M2Time |   0.000000
    unused |   0.000000
    unused |   0.000000
    unused |   0.000000
```

7. Running Bock's nominal scaling model: SIM3 Data: N=67, n=5, m=3:

```
  print "SIM3FR: nobs=67, nvar=5, m=3 cats and Frequency: Samejima";
  options NOECHO;
  sim3fr = [
#include "..\\tdata\\sim3fr.dat"
          ];
  options ECHO;
  sim3fr = shape(sim3fr,.,6);
  cnam = [ "v1":"v5" "wgt" ];
  sim3fr = cname(sim3fr,cnam);

  /* create 7 level ability variable */
  srand(123);
  rcol = round(6. * rand(67,1));
  /* print "rcol[50]=", rcol[ 1:50 ]; */
  sim3f2 = sim3fr -> rcol;
  print "nrow(sim3f2)=", nrow(sim3f2);
  print "ncol(sim3f2)=", ncol(sim3f2);
  /* print "Attached Ability Column=", sim3f2[1:30,]; */


  /* nitr= */
  optn = [ "meth"      "nom" ,
           "nrcat"         3 ,
           "frequ"         6 ,
           "abil"          7 ,
           "tech"   "nrridg" ,
           "phis"          3 ,
           "pres"            ,
           "print"         3 ];
  < gof,est,cov,resi > = irtml(sim3f2,optn);
```

```
**************************************************
IRT ML Estimation of Nominal Categories (Bock, 1972)
**************************************************

Number Rows of Matrix . . . . . .          67
Number Columns of Matrix  . . . . . .       7
Number of Parameters. . . . . .            30
Number of Item Categories . . . . . .       3
Total Number of Categories. . . . . .      15
Maximum Item Level. . . . . .               2
Column of Frequency Variable. . . . . .     6
Total Frequency . . . . . .               100
Number Uni- and Bivariate Moments . . . . . .  50
Number of Theta Points on Scale . . . . . .    7

            ****************************
            Frequencies of Data Categories
            ****************************

          Variable Level Frequency Percentage
          Var_1        0        42    42.00 %
                       1        13    13.00 %
                       2        45    45.00 %
          Var_2        0        38    38.00 %
                       1        25    25.00 %
                       2        37    37.00 %
          Var_3        0        40    40.00 %
                       1        16    16.00 %
                       2        44    44.00 %
          Var_4        0        39    39.00 %
                       1        13    13.00 %
                       2        48    48.00 %
          Var_5        0        43    43.00 %
                       1        23    23.00 %
                       2        34    34.00 %

             Initial Values for Thresholds

                        1            2            3
      Var_1   0.000000000  0.000000000  0.000000000
      Var_2   0.000000000  0.000000000  0.000000000
      Var_3   0.000000000  0.000000000  0.000000000
      Var_4   0.000000000  0.000000000  0.000000000
      Var_5   0.000000000  0.000000000  0.000000000
```

Initial Values for Slopes

	1	2	3
Var_1	1.000000000	1.000000000	1.000000000
Var_2	1.000000000	1.000000000	1.000000000
Var_3	1.000000000	1.000000000	1.000000000
Var_4	1.000000000	1.000000000	1.000000000
Var_5	1.000000000	1.000000000	1.000000000

Theta Scale

	1
Grp_1	-3.000000000
Grp_2	-2.000000000
Grp_3	-1.000000000
Grp_4	0.000000000
Grp_5	1.000000000
Grp_6	2.000000000
Grp_7	3.000000000

Ability Group Frequencies

	1
Grp_1	4.000000000
Grp_2	17.00000000
Grp_3	15.00000000
Grp_4	18.00000000
Grp_5	16.00000000
Grp_6	26.00000000
Grp_7	4.000000000

Frequencies for Ability Groups and Levels

	Var_1_1	Var_1_2	Var_1_3	Var_2_1
Grp_1	1.000000000	0.000000000	3.000000000	0.000000000
Grp_2	9.000000000	2.000000000	6.000000000	9.000000000
Grp_3	6.000000000	3.000000000	6.000000000	6.000000000
Grp_4	6.000000000	4.000000000	8.000000000	2.000000000
Grp_5	5.000000000	4.000000000	7.000000000	6.000000000
Grp_6	13.00000000	0.000000000	13.00000000	12.00000000
Grp_7	2.000000000	0.000000000	2.000000000	3.000000000

Frequencies for Ability Groups and Levels

	Var_2_2	Var_2_3	Var_3_1	Var_3_2
Grp_1	0.000000000	4.000000000	0.000000000	3.000000000
Grp_2	2.000000000	6.000000000	9.000000000	4.000000000
Grp_3	4.000000000	5.000000000	5.000000000	2.000000000
Grp_4	4.000000000	12.00000000	6.000000000	3.000000000
Grp_5	6.000000000	4.000000000	7.000000000	2.000000000
Grp_6	9.000000000	5.000000000	12.00000000	2.000000000
Grp_7	0.000000000	1.000000000	1.000000000	0.000000000

Frequencies for Ability Groups and Levels

	Var_3_3	Var_4_1	Var_4_2	Var_4_3
Grp_1	1.000000000	0.000000000	0.000000000	4.000000000
Grp_2	4.000000000	5.000000000	2.000000000	10.00000000
Grp_3	8.000000000	6.000000000	2.000000000	7.000000000
Grp_4	9.000000000	8.000000000	2.000000000	8.000000000
Grp_5	7.000000000	6.000000000	6.000000000	4.000000000
Grp_6	12.00000000	11.00000000	1.000000000	14.00000000
Grp_7	3.000000000	3.000000000	0.000000000	1.000000000

Frequencies for Ability Groups and Levels

	Var_5_1	Var_5_2	Var_5_3
Grp_1	0.000000000	3.000000000	1.000000000
Grp_2	4.000000000	2.000000000	11.00000000
Grp_3	5.000000000	6.000000000	4.000000000
Grp_4	4.000000000	3.000000000	11.00000000
Grp_5	9.000000000	5.000000000	2.000000000
Grp_6	20.00000000	4.000000000	2.000000000
Grp_7	1.000000000	0.000000000	3.000000000

------ ML Estimation for Item: 1 ------
Newton-Raphson Ridge Optimization
Without Parameter Scaling
User Specified Gradient
User Specified Hessian (dense)

Iteration Start:

N. Variables	4		
N. Bound. Constr.	8	N. Mask Constr.	0
Criterion	109.8612289	Max Grad Entry	20.33333333
N. Active Constraints	0		

```
Iter rest nfun act   optcrit  difcrit maxgrad   ridge      rho
   1    0    2   0  99.06964 10.79159 3.77351 0.00000  1.12135
   2    0    3   0  98.41439 0.655247 0.48570 0.00000  1.07747
   3    0    4   0  98.40279 0.011598 0.01373 0.00000  1.01628
   4    0    5   0  98.40278 7.2e-006  9e-006 0.00000  1.00044

 Successful Termination After     4 Iterations
 ABSGCONV convergence criterion satisfied.
 Criterion       98.40278453        Max Grad Entry  9.1492e-006
 N. Active Constraints     0        Ridge (lambda)  0.000000000
 Act.dF/Pred.dF  1.000438671
 N. Function Calls         6        N. Gradient Calls         3
 N. Hessian Calls          6        Preproces. Time           0
 Time for Method           0        Effective Time            0


                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

        Parameter     Estimate   Gradient   Active BC

         1 ICPT_1_1 1.17199405 -5.72e-006
         2 ICPT_1_2-0.06630391  3.00e-006
         3 ICPT_2_1 0.17187717 -9.15e-006
         4 ICPT_2_2-0.01065395  4.82e-006

         Value of Objective Function =      98.4028


           ------ ML Estimation for Item: 2 ------
              Newton-Raphson Ridge Optimization
                  Without Parameter Scaling
                   User Specified Gradient
                User Specified Hessian (dense)


 Iteration Start:
 N. Variables              4
 N. Bound. Constr.         8        N. Mask  Constr.          0
 Criterion       109.8612289        Max Grad Entry  18.33333333
 N. Active Constraints     0
```

```
Iter rest nfun act   optcrit  difcrit maxgrad   ridge      rho
   1    0    2   0  105.2185 4.642683 1.88824 0.00000  1.00804
   2    0    3   0  105.1349 0.083631 0.06013 0.00000  1.00126
   3    0    4   0  105.1348 7.0e-005  3e-005 0.00000  1.00023
   4    0    5   0  105.1348 2.6e-011  2e-011 0.00000  1.00093

 Successful Termination After     4 Iterations
 GCONV convergence criterion satisfied.
 Criterion       105.1348447        Max Grad Entry  2.4929e-011
 N. Active Constraints     0        Ridge (lambda)  0.000000000
 Act.dF/Pred.dF  1.000927036
 N. Function Calls         6        N. Gradient Calls         6
 N. Hessian Calls          6        Preproces. Time           0
 Time for Method           0        Effective Time            0


                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

        Parameter     Estimate   Gradient   Active BC

         1 ICPT_1_1 0.46995132 -1.53e-011
         2 ICPT_1_2 0.01660072  1.12e-011
         3 ICPT_2_1-0.09878533  2.49e-011
         4 ICPT_2_2 0.27131668 -1.97e-011

         Value of Objective Function =      105.135
```

For space reasons we skip the iteration history for items 3 and 4.

```
            ------ ML Estimation for Item: 5 ------
               Newton-Raphson Ridge Optimization
                   Without Parameter Scaling
                    User Specified Gradient
                User Specified Hessian (dense)


 Iteration Start:
 N. Variables              4
 N. Bound. Constr.         8        N. Mask  Constr.          0
```

```
 Criterion       109.8612289        Max Grad Entry  20.33333333
 N. Active Constraints     0


Iter rest nfun act   optcrit  difcrit maxgrad   ridge      rho
   1    0    2   0  99.24250 10.61873 2.06305 0.00000  0.95052
   2    0    3   0  99.15265 0.089847 0.08080 0.00000  0.98379
   3    0    4   0  99.15258 7.2e-005  7e-005 0.00000  1.00061
   4    0    5   0  99.15258 6.8e-011  7e-011 0.00000  1.00002

 Successful Termination After     4 Iterations
 GCONV convergence criterion satisfied.
 Criterion       99.15257844        Max Grad Entry  6.7541e-011
 N. Active Constraints     0        Ridge (lambda)  0.000000000
 Act.dF/Pred.dF  1.000016241
 N. Function Calls         6        N. Gradient Calls        15
 N. Hessian Calls          6        Preproces. Time           0
 Time for Method           0        Effective Time            0


                    ********************
                    Optimization Results
                    ********************

                    Parameter Estimates
                    -------------------

        Parameter     Estimate   Gradient   Active BC

         1 ICPT_1_1 0.45933792  9.02e-012
         2 ICPT_1_2 0.08852700 -3.47e-011
         3 ICPT_2_1 0.48338879  1.45e-011
         4 ICPT_2_2 0.54298112 -6.75e-011

         Value of Objective Function =      99.1526
```

The following are the individual ability estimates:

```
          ML Estimates of Subject Abilities

                              1
                    1  6.851977853
                    2  0.686270002
                    3  3.928418199
                    4  2.878465635
                    5 -1.098448410
```

```
 6 -1.555362091
 7  1.175133917
 8 -2.564951050
 9  0.028434059
10 -0.632285265


60 -8.402857976
61  3.488439596
62 -1.144551485
63  1.965857333
64 -2.294090448
65  1.219449729
66 -2.520463933
67 -3.014899680
```

GOF=

```
                              gof
                              ***

                    Sparse Column Vector gof

     C |     NegLike    DetHess         G2       G2DF     G22PRB
          -497.95165     .       198.30516     .          .

     C |      MAXMEM     SUMMEM
           2560.0000  6652.0000
```

3.3 Function `irtms`

< gof,scal,loev,pairs > = irtms(data,optn)

Purpose: This function fits a nonparametric IRT model which is known as Mokken scaling. The algorithm is an forward stepwise process adding item by item to a scale as long as Loevinger's H index is larger than a lower threshold c which can be specified and is by default set to $c = .3$. The greedy algorithm is therefore fast but suboptimal. It is based on a paper by Hardouin (2007). The algorithm is designed for binary data. (We do not agree with the author's claim that the data could also be polytomous.) This function is based on the excellent SAS/IML code written by J.-B. Hardouin.

Input: There are only two input arguments:

data This is an $m \times n$ matrix

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.

Option	Second Column	Meaning
`"adj"`		
`"cval"`	real	default=0.3
`"kern"`	int	for $k > 0$ chooses first k items as seed for first scale; default $k = 0$
`"meth"`	char	`"mscal"`: performs Mokken scaling `"loevc"`: returns Guttman errors and Loevinger's H
`"list"`		listwise treatment of missing values observations with missing values are deleted
`"orig"`		intrascale listwise treatment of missing values this is Hardouin's original approach
`"pair"`		pairwise treatment of missing values
`"print"`	int	specifies amount of printed output; =0: no output is printed; default=2

Output: The function `irtms` returns four arguments

gof a vector of scalar results

scal only nonmissing for `"meth" "mscal"`: a membership vector with integers specifying to which scale the item belongs

loev returns a table of Guttman errors and Loevinger H (for items in scales)

pairs only nonmissing for `"meth" "loevc"`: returns a table of pairwise Guttman errors and Loevinger H coefficients

Restrictions: 1. The input data set cannot have string values and must contain enough nonmissing observations.

2. The data must be $(0, 1)$ binary or missing.

Relationships: irtml(), noharm()

Examples: 1. Guttman errors and Loevinger's H for subset of items:

```
options NOECHO;
sip1 = [
#include "..\\tdata\\sip1.dat"
       ];
options ECHO;
sip2 = shape(sip1,.,19);
sip3 = sip2[,10:19];
cnam = [" m1:10 "];
sip3 = cname(sip3,cnam);


/* all listwise */
optn = [ "meth"  "loevc" ,
         "adj"           ,
         "list"          ,
         "print"       3 ];
< gof,scal,loevh,pairs > = irtms(sip3,optn);
```

```
                ******************************
                Mokken Scaling (Hardouin, 2007)
                ******************************

Number Rows of Matrix . . . . . .          483
Number Columns of Matrix  . . . . .         10
Listwise Missing Value: Nobs. . . . .      466
Adjusted Divisor. . . . . .
```

```
                 Guttman Errors and Loevinger H
                 ******************************

                     Dense Matrix (11 by 7)

     |        Nobs    Easyness   ObsGutEr   ExpGutEr   LoevingH
-----------------------------------------------------------------
```

m1	466.00000	0.2081545	294.00000	426.61159	0.3108485
m2	466.00000	0.1266094	207.00000	310.68455	0.3337293
m3	466.00000	0.5042918	423.00000	486.41416	0.1303707
m4	466.00000	0.1888412	283.00000	405.46352	0.3020334
m5	466.00000	0.2854077	347.00000	478.68026	0.2750902
m6	466.00000	0.6351931	218.00000	410.04292	0.4683483
m7	466.00000	0.0429185	86.000000	119.91416	0.2828203
m8	466.00000	0.3583691	317.00000	498.96352	0.3646830
m9	466.00000	0.5901288	236.00000	448.30258	0.4735698
m10	466.00000	0.1072961	201.00000	273.00429	0.2637478
Scale	466.00000	.	1306.0000	1929.0408	0.3229796

	zStatist	pValue
m1	12.109114	0.0000000
m2	11.362900	0.0000000
m3	4.8359389	6.63e-007
m4	11.556662	0.0000000
m5	10.947751	0.0000000
m6	15.117751	0.0000000
m7	5.9708011	3.78e-009
m8	14.371111	0.0000000
m9	16.414026	0.0000000
m10	8.4399812	0.0000000
Scale	25.162891	0.0000000

Pairwise Guttman Errors and Loevinger H

****

Dense Matrix (45 by 7)

	Nobs	Easyness	ObsGutEr	ExpGutEr	LoevingH
1	466.00000	0.0793991	22.000000	46.718884	0.5290983
2	466.00000	0.0965665	52.000000	48.083691	-0.0814478
3	466.00000	0.0708155	26.000000	29.246781	0.1110133
4	466.00000	0.0708155	55.000000	69.682403	0.2107046
5	466.00000	0.0472103	37.000000	47.858369	0.2268855
6	466.00000	0.1459227	20.000000	43.622318	0.5415191
7	466.00000	0.1223176	40.000000	69.315451	0.4229281
8	466.00000	0.0579399	32.000000	42.160944	0.2410037
9	466.00000	0.1351931	70.000000	65.929185	-0.0617453
10	466.00000	0.0708155	55.000000	62.884120	0.1253754
11	466.00000	0.1716738	17.000000	35.386266	0.5195876
12	466.00000	0.1137339	6.0000000	21.523605	0.7212363

13	466.00000	0.3519313	71.000000	85.729614	0.1718148
14	466.00000	0.1587983	14.000000	32.103004	0.5639037
15	466.00000	0.2296137	26.000000	48.519313	0.4641309
16	466.00000	0.0214592	10.000000	15.836910	0.3685637
17	466.00000	0.0128755	14.000000	17.467811	0.1985258
18	466.00000	0.0214592	10.000000	9.9141631	-0.0086580
19	466.00000	0.0107296	15.000000	16.223176	0.0753968
20	466.00000	0.0150215	13.000000	14.291845	0.0903904
21	466.00000	0.0343348	4.0000000	7.2961373	0.4517647
22	466.00000	0.1201717	41.000000	62.238197	0.3412406
23	466.00000	0.0686695	27.000000	37.856223	0.2867751
24	466.00000	0.1952790	76.000000	82.783262	0.0819400
25	466.00000	0.1094421	37.000000	56.463519	0.3447096
26	466.00000	0.1673820	55.000000	85.336910	0.3554958
27	466.00000	0.2982833	28.000000	60.922747	0.5404015
28	466.00000	0.0321888	5.0000000	12.832618	0.6103679
29	466.00000	0.1652361	20.000000	39.757511	0.4969504
30	466.00000	0.1008584	12.000000	24.182403	0.5037714
31	466.00000	0.3347639	79.000000	96.319742	0.1798151
32	466.00000	0.1630901	12.000000	36.068670	0.6673013
33	466.00000	0.2231760	29.000000	54.512876	0.4680156
34	466.00000	0.4957082	44.000000	100.32189	0.5614118
35	466.00000	0.0364807	3.0000000	8.1974249	0.6340314
36	466.00000	0.2982833	28.000000	68.448498	0.5909333
37	466.00000	0.0278970	37.000000	39.592275	0.0654743
38	466.00000	0.0407725	31.000000	43.669528	0.2901229
39	466.00000	0.0665236	19.000000	24.785408	0.2334199
40	466.00000	0.0257511	38.000000	40.557940	0.0630688
41	466.00000	0.0493562	27.000000	35.729614	0.2443243
42	466.00000	0.0901288	8.0000000	18.240343	0.5614118
43	466.00000	0.0171674	12.000000	17.854077	0.3278846
44	466.00000	0.0643777	20.000000	32.081545	0.3765886
45	466.00000	0.0879828	9.0000000	20.493562	0.5608377

	zStatist	pValue
1	8.4726447	0.0000000
2	-0.8927922	0.8140158
3	0.9036542	0.1830894
4	4.2758579	9.52e-006
5	3.8607044	5.65e-005
6	5.5860876	1.16e-008
7	7.3988118	2.18e-013
8	3.1309441	8.71e-004
9	-0.8342687	0.7979352

```
      10 |   2.0640987  0.0195042
      11 |   4.3534801  6.70e-006
      12 |   4.4875723  3.60e-006
      13 |   2.8320017  0.0023129
      14 |   4.4463689  4.37e-006
      15 |   4.7934619  8.20e-007
      16 |   3.2825693  5.14e-004
      17 |   2.3810154  0.0086325
      18 |  -0.0391980  0.5156337
      19 |   0.7135619  0.2377491
      20 |   0.6531194  0.2568396
      21 |   1.5633814  0.0589815
      22 |   5.0482071  2.23e-007
      23 |   3.1504611  8.15e-004
      24 |   1.3092338  0.0952276
      25 |   4.7990215  7.97e-007
      26 |   6.4824917  1.44e-010
      27 |   6.6000057  6.56e-011
      28 |   3.7294354  9.60e-005
      29 |   4.5789145  2.34e-006
      30 |   3.4469829  2.83e-004
      31 |   3.2593490  5.58e-004
      32 |   5.7862103  1.15e-008
      33 |   5.3154598  5.32e-008
      34 |   11.008706  0.0000000
      35 |   2.4128769  0.0079136
      36 |   7.9366628  3.33e-015
      37 |   0.9546907  0.1698671
      38 |   5.6966291  1.96e-008
      39 |   1.7301142  0.0418049
      40 |   0.9772008  0.1642349
      41 |   2.8901995  0.0019250
      42 |   3.1807163  7.35e-004
      43 |   4.3187369  7.85e-006
      44 |   3.7671193  8.26e-005
      45 |   3.4942369  2.38e-004
```

```
    print "IRTMS: gof=",gof;
    print "IRTMS: scal=",scal;
    print "IRTMS: LoevH=",loevh;
    print "IRTMS: pairs=",pairs;
```

```
IRTMS: gof=
           |         1
-----------------------
```

```
 Failure |   0.00000
    Time |   0.00000
LoevingH |   0.32298
    Nobs |    466.00
ObsGuttE |    1306.0
ExpGuttE |    1929.0
zStatist |    25.163
  pValue |   0.00000
  unused |   0.00000
  unused |   0.00000
```

The second result `scal` is missing.

2. Mokken scaling with intrascale listwise treatment of missing values:

```
optn = [ "meth"  "mscal" ,
         "adj"           ,
         "orig"          ,
         "print"       3 ];
< gof,scal,loevh > = irtms(sip2,optn);


              ******************************
              Mokken Scaling (Hardouin, 2007)
              ******************************

Number Rows of Matrix . . . . . .          483
Number Columns of Matrix  . . . . . .       19
Listwise Treatment of Missing Values. . . . . .
C Parameter . . . . . .                 0.3000
Adjusted Divisor. . . . . .


     ***************** Create Scale 1 ****************
          First item pair of the scale (c8,c9).
                  Item m4 is excluded.
                  Item m7 is excluded.
                  Item m10 is excluded.
          Item c7 is selected in the sub-scale 1.
                  Item m3 is excluded.
          Item c2 is selected in the sub-scale 1.
                  Item m5 is excluded.
                  Item m6 is excluded.
                  Item m9 is excluded.
          Item c3 is selected in the sub-scale 1.
          Item c4 is selected in the sub-scale 1.
          Item c6 is selected in the sub-scale 1.
```

```
     Item c5 is selected in the sub-scale 1.
     All the Hj coefficients are too small.
No other item can be selected in the sub-scale 1.


**************** Create Scale 2 ****************
      First item pair of the scale (m2,m6).
              Item m3 is excluded.
              Item m7 is excluded.
     Item m9 is selected in the sub-scale 2.
     Item m1 is selected in the sub-scale 2.
              Item m10 is excluded.
     Item m8 is selected in the sub-scale 2.
     Item m5 is selected in the sub-scale 2.
     Item m4 is selected in the sub-scale 2.
     All the Hj coefficients are too small.
No other item can be selected in the sub-scale 2.


**************** Create Scale 3 ****************
     First item pair of the scale (c1,m10).
              Item m3 is excluded.
     Item m7 is selected in the sub-scale 3.
       There are no more items remaining.

**************** Create Scale 4 ****************
There is no more item or only one item remaining.


    Guttman Errors and Loevinger H for Scale 1

              Nobs     Easyness     ObsGutEr     ExpGutEr
c8      472.0000000  0.197033898  223.0000000  393.3114407
c9      472.0000000  0.527542373  217.0000000  421.9046610
c7      472.0000000  0.349576271  278.0000000  508.4639831
c2      472.0000000  0.137711864  176.0000000  306.6843220
c3      472.0000000  0.148305085  183.0000000  326.0169492
c4      472.0000000  0.315677966  307.0000000  504.5317797
c6      472.0000000  0.302966102  324.0000000  497.3368644
c5      472.0000000  0.440677966  322.0000000  481.4067797
Scale   483.0000000            .  1015.000000  1719.828390


    Guttman Errors and Loevinger H for Scale 1

             LoevingH     zStatist       pValue
  c8      0.433019290  16.66674070  0.000000000
```

c9	0.485665791	16.52407217	0.000000000
c7	0.453255276	19.29739488	0.000000000
c2	0.426119996	14.55731593	0.000000000
c3	0.438679491	15.48762561	0.000000000
c4	0.391515040	16.89659869	0.000000000
c6	0.348530095	14.96911338	0.000000000
c5	0.331126994	12.91312680	0.000000000
Scale	0.409824837	31.68242975	0.000000000

Guttman Errors and Loevinger H for Scale 2

	Nobs	Easyness	ObsGutEr	ExpGutEr
m2	470.0000000	0.125531915	136.0000000	220.1829787
m6	470.0000000	0.636170213	136.0000000	300.5234043
m9	470.0000000	0.589361702	146.0000000	326.2212766
m1	470.0000000	0.208510638	195.0000000	325.8595745
m8	470.0000000	0.359574468	218.0000000	374.2468085
m5	470.0000000	0.285106383	237.0000000	365.4595745
m4	470.0000000	0.189361702	210.0000000	307.8212766
Scale	483.0000000	.	639.0000000	1110.157447

Guttman Errors and Loevinger H for Scale 2

	LoevingH	zStatist	pValue
m2	0.382331910	10.57301467	0.000000000
m6	0.547456211	14.99380589	0.000000000
m9	0.552451019	16.13953370	0.000000000
m1	0.401582721	13.71691950	0.000000000
m8	0.417496703	14.26826307	0.000000000
m5	0.351501461	12.32240045	0.000000000
m4	0.317785949	10.57994730	0.000000000
Scale	0.424405969	24.94364907	0.000000000

Guttman Errors and Loevinger H for Scale 3

	Nobs	Easyness	ObsGutEr	ExpGutEr
c1	475.0000000	0.515789474	21.00000000	35.83157895
m10	475.0000000	0.109473684	27.00000000	44.77052632
m7	475.0000000	0.046315789	20.00000000	30.24421053
Scale	483.0000000	.	34.00000000	55.42315789

Guttman Errors and Loevinger H for Scale 3

```
                    LoevingH     zStatist       pValue
          c1      0.413924794  3.614150509  0.000150667
          m10     0.396924668  4.811738911  7.4811e-007
          m7      0.338716414  3.791337268  7.4919e-005
          Scale   0.386538023  4.929030118  4.1319e-007
```

```
    print "IRTMS: gof=",gof;
    print "IRTMS: scal=",scal;
    print "IRTMS: LoevH=",loevh;
```

```
IRTMS: gof=
           |         1
-----------------------
   Failure |   0.00000
      Time |   1.00000
  N Scales |    3.0000
    unused |   0.00000
    unused |   0.00000
    unused |   0.00000
    unused |   0.00000
    unused |   0.00000
    unused |   0.00000
    unused |   0.00000
```

```
IRTMS: scal=
   |        c1        c2        c3        c4        c5
-------------------------------------------------------
 1 |    3.0000   1.00000   1.00000   1.00000   1.00000

   |        c6        c7        c8        c9        m1
-------------------------------------------------------
 1 |   1.00000   1.00000   1.00000   1.00000    2.0000

   |        m2        m3        m4        m5        m6
-------------------------------------------------------
 1 |    2.0000    .         2.0000    2.0000    2.0000

   |        m7        m8        m9       m10
-----------------------------------------------
 1 |    3.0000    2.0000    2.0000    3.0000
```

```
IRTMS: LoevH=
        |       Nobs   Easyness   ObsGutEr   ExpGutEr
```

c8	472.00	0.19703	223.00	393.31
c9	472.00	0.52754	217.00	421.90
c7	472.00	0.34958	278.00	508.46
c2	472.00	0.13771	176.00	306.68
c3	472.00	0.14831	183.00	326.02
c4	472.00	0.31568	307.00	504.53
c6	472.00	0.30297	324.00	497.34
c5	472.00	0.44068	322.00	481.41
Scale	483.00	.	1015.0	1719.8
m2	470.00	0.12553	136.00	220.18
m6	470.00	0.63617	136.00	300.52
m9	470.00	0.58936	146.00	326.22
m1	470.00	0.20851	195.00	325.86
m8	470.00	0.35957	218.00	374.25
m5	470.00	0.28511	237.00	365.46
m4	470.00	0.18936	210.00	307.82
Scale	483.00	.	639.00	1110.2
c1	475.00	0.51579	21.000	35.832
m10	475.00	0.10947	27.000	44.771
m7	475.00	0.05	20.000	30.244
Scale	483.00	.	34.000	55.423

	LoevingH	zStatist	pValue
c8	0.43302	16.667	0.00000
c9	0.48567	16.524	0.00000
c7	0.45326	19.297	0.00000
c2	0.42612	14.557	0.00000
c3	0.43868	15.488	0.00000
c4	0.39152	16.897	0.00000
c6	0.34853	14.969	0.00000
c5	0.33113	12.913	0.00000
Scale	0.40982	31.682	0.00000
m2	0.38233	10.573	0.00000
m6	0.54746	14.994	0.00000
m9	0.55245	16.140	0.00000
m1	0.40158	13.717	0.00000
m8	0.41750	14.268	0.00000
m5	0.35150	12.322	0.00000
m4	0.31779	10.580	0.00000
Scale	0.42441	24.944	0.00000
c1	0.41392	3.6142	2e-004
m10	0.39692	4.8117	7e-007
m7	0.33872	3.7913	7e-005

```
Scale |    0.38654     4.9290     4e-007
```

3.4 Function `mds`

< gof,conf,wgt,eval,evec > = mds(data,optn<,xini>)

< gof,conf,wgt,.,grad > = mds(data,optn<,xini>)

Purpose: This function implements a variety of *metric* and *nonmetric* methods for 2-way and 3-way multidimensional scaling. Similar to cluster analysis, multidimensional scaling analyzes the relations between the rows of a data set (observations, cases) and is therefore different of most statistical methods which anayze the relations between the columns (variables, items) of a data set.

The result is usually a $n \times d$ configuration matrix $\hat{\mathbf{X}}$ of n points in a small $d << p$ dimensional space, whose interpoint distances $d(x_i, x_j)$ fit in some way the distances among the larger p dimensional space of the input data. If $d = 2$ or $d = 3$, the plot of the $n \times d$ result confugration may uncover the most important relations among the observations of the data, such as clusters of similar subjects.

The following estimation methods are implemented:

COSPA 3-way COmmon SPace Analysis by Schoenemann (1972)

INDSCAL 3-way (Metric) INdividual Differences Scaling by Carroll & Chang (1970) using the Canonical Decomposition (CANDECOMP) technique is based on Horan's (1969) model

SUMSCAL 3-way (Metric) individual differences scaling by de Leeuw & Pruzansky (1978)

KYST 2-way Kruskal-Young-Shepard-Torgerson metric and nonmetric scaling; see Young (1987)

Note, that this function is still in the works and will be extended in the next newsletter.

Input: data This function permits the following forms of data input:

1. $n \times p$ matrices or $K \times n \times p$ 3-way tables (tensors) of *raw* data.
2. symmetric $n \times n$ matrices or $K \times n \times n$ 3-way tables (tensors) of distance (dissimilarity) data.
3. symmetric $n \times n$ matrices or $K \times n \times n$ 3-way tables (tensors) of correlation, covariance, or scalar product (similarity) 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.

xini an $n \times d$ matrix of initial values for the $\mathbf{X}$ configuration of points

Option	Second Column	Meaning
"addc"		add constant if distance data do not satisfy triangle inequality
"dimmax"	int	maximum number dimensions d (def=2)
"dimmin"	int	minimum number dimensions d (def=2)
"dimdif"	int	step size dimensions d (def=1)
"data"	string	kind of input data
	"raw"	$n \times p$ raw data
	"diss"	dissimilarities
	"simi"	similarities
	"dist"	distances
	"corr"	correlations
	"cova"	covariances
	"scp"	scalar product data
"gutlin"		apply Guttman-Lingoes transformation
"met"	string	estimation method
	"torg"	classic Torgerson
	"cos"	COSPA
	"sums"	SUMSCAL
	"inds"	INDSCAL
	"kyst"	KYST
"maxit"	int	number of iterations
"ndim"	int	number dimensions d (def=2)
"nopr"		do not print
"pdis"		print distance matrices
"pcoef"		print X and W configurations
"phis"		print iteration history
"print"	int	controls printed output
"pres"		printed observationwise output
"seed"	int	seed of random generator (def: time of day)
"sing"	real	singularity criterion (def=1.e-8)
"xtol"	real	termination criterion (def=1.e-5)

Specific options for the INDSCAL, COSPA, and SUMSCAL methods:

This three methods are specifically designed for the 3-way metric MDS of a set of K scalar product matrices. For input raw data the `mds` function will produce distance or scalar product matrices which are suitable. They only fit 3-way scalar product data in a least squares sense and are therefore rather sensitive to outliers of the data. All other input data types, like (squared) Euclidean distances, are transformed internally to scalar product data.

Option	Second Column	Meaning
"dist"	string	form of distance function
	"euc"	use Euclidean distances (def)
	"squ"	use squared Euclidean distances
"norm"		use normed scalar products
"plin"	int	print only selected steps of INDSCAL iteration

Specific options for the more general KYST method:

These three methods use more general algorithms and are applicable to a wider range of input data. They also permit to fit city-block distances and nonlinear or monotone regression between the data and the model distances. These methods can also deal with a reasonable amount of missing data.

Option	Second Column	Meaning
"acsav"	real	for termination test (def=0.66)
"cutof"	real	threshold for removing small distances from fit
"cosav"	real	for termination test (def=0.66)
"coord"	char	type of normalization
	"as-is"	no normalization
	"stand"	normalize at each iteration
	"rotat"	rotate to principal components (def)
"diag"	char	do or don't use diagonal of data matrix
	"absent"	do not use diagonal
	"present"	use diagonal (def)
"dpow"	real	power of diatance function (def=2.)
		should be in [1., 4.]
"ftol"	real	termination test for stress value (def=.01)
"gtol"	real	termination test for 2-norm of gradient (def=0)
"pre-it"	int	number of iterations for start estimates
"plotco"	char	configuration plotting (obs vs. dim)
	"non"	
	"som"	
	"all"	
"plotsc"	char	scatter plotting (data vs. model)
	"non"	
	"som"	
	"all"	
	"joi"	

Option	Second Column	Meaning
"poldeg"	int	polynomial degree of regression
		should be in $[1, 4]$
"regres"	char	type of regression between data and model values
	"asc"	isotone ascending (def)
	"des"	isotone descending
	"poly"	polynomial with intercept
	"poln"	polynomial without intercept
"split"	char	split data values in groups (for unfolding)
	"byrow"	
	"bygroup"	
	"bydeck"	
"sfrm"	int	kind of stress formula for optimization
		=1: default, =2: for unfolding
"sratst"	real	for termination test (def=.999)
"start"	char	method for initial estimates
	"rand"	random
	"tors"	Torgerson-Young quasi nonmetric
"ties"	char	type of tie processing in isotone regression
	"prim"	type 1 processing
	"seco"	type 2 processing
"unsym"		input of unsymmetric data matrix
		(does not refer to "data" "raw" input)

Output: Let $s = \sum_{d=dimmin}^{dimmax} d$, i.e. for only one dimensionality $dimmin = dimmax$ there is $s = d$.

GOF a vector with some scalar results

conf this is a $n \times s$ matrix containing all of the computed $n \times d$ configurations horicontally concatenated

wgt the $w \times s$ matrix of the individual weights

eval or [i] for INDSCAL, COSPA, SUMSCAL: this a s vector of eigenvalues depending on the method [ii] for KYST:

evec or grad [i] for INDSCAL, COSPA, SUMSCAL: this is a $n \times s$ matrix of eigenvectors [ii] for KYST: it contains the gradient at the solution

Restrictions: 1.

2.

Relationships: cluster(), dist()

Examples: 1. Randomly generated raw data in tensor form `X[2,n,p]`:

```
srand(123);
nind = [ 2 10 10 ]; m = p = nind[1];
nway = size(nind); n = nind[2]; nd = 4;
```

```
none = cons(n,1,1.); pone = cons(p,1,1.);

/* [1] Uniform random generation of pxnd weight matrix: W[p,nd] */
wght = rand(p,nd,"g");
print "Uniform: Unscaled wght=",wght;


/* [2] Normal random generated nxnd configuration: X[n,nd] */
coef  = rand(n,nd,"g","norm",0.,1.);
coef += rand(n,nd,"g","norm",1.,1.);
mu = cons(nd,1,1.); id = ide(nd);
for (j = 1; j <= n; j++) {
  coef[j,] += mrand("mnor",mu,id)‘;
}
print "Normal: Unscaled coef=",coef;


/* [3] Normalize data[p+n,nd]: NORCONF() */
/* [3.1] center X[n,nd] */
mu = coef[+,] / n;

coef = coef - none * mu;
print "Centered Coef=", coef;

/* [3.2] scaling coef[n,nd] */
tv = coef[**,]; rv = 1. ./ sqrt(tv);
coef = coef .* (none * rv);
print "Scaled Coef=", coef;
print "Mean=", coef[+,];
print "Scale=", coef[**,];

/* [3.3] scaling wght[p,nd] */
twgt = wght .* (pone * tv);
print "Scaled WGT=", twgt;


/* [4] Concatenate COEF and WGHT */
wcof = twgt |> coef;


/* [5] Permute dimensions according weights
       sum of squares in descending order */
wscl = twgt[**,];
wrnk = wscl[>:]; print "Ranks=", wrnk;
wcof = wcof[,wrnk];
n1 = p+1; n2 = p+n;
wght = wcof[1:p,]; coef = wcof[n1:n2,];
```

```
/* [6] Weighted Configuration: Y = X * W */
real wxdat[p,n,nd];
for (j = 1; j <= p; j++) {
  wv = wght[j,];
  wxdat[j,,] = coef .* (none * wv);
  /* print "WXdat[j,,]=", wxdat[j,,]; */
}
print "Tensor WXDAT=",wxdat;
```

```
Tensor WXDAT=
                  ******************************
                  Tensor wxdat with 3 Dimensions
                  ******************************


                           wxdat_1
                           *******

                      Dense Matrix (10 by 4)

              |          1          2          3          4
--------------------------------------------------------------
            1 |  -14.554672 -1.6301023  4.9856908  0.0606599
            2 |  -3.0380098 -4.5080465  1.0150659  0.0439437
            3 |   6.4290345 -7.0272730  0.4288613 -0.6636031
            4 |  -8.4827305  6.9577941 -1.4993646  0.1756061
            5 |   18.784123  1.9255437  3.3502585  0.0904182
            6 |   13.203212  0.4243445 -3.0888729 -0.5465808
            7 |   1.3760075  3.8225599  1.3278726  0.2144417
            8 |  -6.3482351  1.5903091  2.2758083  0.0053984
            9 |  -4.2046344 -5.6754399 -5.2610451  0.3112615
           10 |  -3.1640951  4.1203103 -3.5342749  0.3084545

                           wxdat_2
                           *******

                      Dense Matrix (10 by 4)

              |          1          2          3          4
--------------------------------------------------------------
            1 |  -6.3948545 -0.7757735  3.4352285  0.2674217
            2 |  -1.3348037 -2.1454008  0.6993982  0.1937275
            3 |   2.8247109 -3.3443127  0.2954930 -2.9255223
            4 |  -3.7270388  3.3112474 -1.0330885  0.7741671
            5 |   8.2531390  0.9163754  2.3083869  0.3986123
```

```
     6 |   5.8010665  0.2019476 -2.1282877 -2.4096247
     7 |   0.6045734  1.8191745  0.9149276  0.9453753
     8 |  -2.7892102  0.7568357  1.5680719  0.0237990
     9 |  -1.8473810 -2.7009689 -3.6249525  1.3722094
    10 |  -1.3902016  1.9608753 -2.4351774  1.3598348
```

(a) Analysis with COSPA:

```
optn = [ "data"      "raw" ,
         "meth"    "cospa" ,
         "ndim"          2 ,
         "norm"           ];
< gof,conf,wgt,eval,evec > = mds(wxdat,optn);
print "GOF=",gof;
print "X Configuration=",conf;
print "W Configuration=",wgt;
print "Eigenvalues=",eval;
print "Eigenvectors=",evec;
```

```
                ****************************
                3-Way Multidimension Scaling
                ****************************

                        Raw Input Data
 Number Rows of Matrix . . . . . .          10
 Dimension . . . . . .           2
 Amount Input Information. . . . . .        90
 Number of Parameters. . . . . .            24
 Degrees of Freedom. . . . . .              66

                 *************************
                 Compute SCHOENEMANN COSPA
                 *************************

Eigenvalues of Mean Scalar Product Matrix
-----------------------------------------

  1 :      0.9573     0.2185     0.1664    0.02963  1.191e-016
  6 :   1.633e-017 -1.537e-017 -4.755e-017 -1.052e-016 -3.128e-016

             Unrotated common space configuration

                      DIM_1        DIM_2
               O_01  -0.465592568  0.071152585
               O_02  -0.093331284  0.147963476
               O_03   0.227242154  0.246678538
```

```
O_04  -0.278491720 -0.215828327
O_05   0.585362887 -0.105058979
O_06   0.434267720 -0.008289568
O_07   0.032286775 -0.137844258
O_08  -0.205208492 -0.042378202
O_09  -0.128315170  0.181828134
O_10  -0.108220302 -0.138223400

Undiagonalized Weight Matrices:
*******************************

      Weight matrix:    1

              DIM_1        DIM_2
DIM_1   1.017745116
DIM_2  -0.027635222  0.907661656

      Weight matrix:    2

              DIM_1        DIM_2
DIM_1   0.982254884
DIM_2   0.027635222  1.092338344

   Orthogonal Rotation Matrix

              DIM_1        DIM_2
DIM_1   0.230560993  0.973057875
DIM_2   0.973057875 -0.230560993

Rotated common space configuration

             DIM_1        DIM_2
O_01  -0.038111902 -0.469453525
O_02   0.122458472 -0.124931347
O_03   0.292425671  0.164245319
O_04  -0.274222781 -0.221226968
O_05   0.032733382  0.593814469
O_06   0.092058967  0.424478876
O_07  -0.126686370  0.063198410
O_08  -0.088549516 -0.189908979
O_09   0.147344825 -0.166780562
O_10  -0.159450748 -0.073435693

 Diagonalized Weight Matrices:
 *****************************
```

```
      Weight matrix:    1

              DIM_1        DIM_2
DIM_1   0.901113634
DIM_2  -3.8580e-015  1.024293137

      Weight matrix:    2

              DIM_1        DIM_2
DIM_1   1.098886366
DIM_2   2.1927e-015  0.975706863

Schoenemann's Testindizes for COSPA

  Matrix  CommonSpace  Diagonality
       1  0.964379639  2.8395e-015
       2  0.885697342  1.4974e-015

  ******************************
  Result Normalized Configuration
  ******************************

     Common Space Configuration

             DIM_1        DIM_2
O_01  -0.489977791 -0.075071833
O_02  -0.130393281  0.241215514
O_03   0.171426039  0.576012484
O_04  -0.230898897 -0.540156903
O_05   0.619775732  0.064477364
O_06   0.443036873  0.181335360
O_07   0.065961412 -0.249543518
O_08  -0.198211702 -0.174422535
O_09  -0.174072122  0.290236005
O_10  -0.076646263 -0.314081938

     Subjective Weight Matrices

             DIM_1        DIM_2
 W_1   0.940278879  0.232245207
 W_2   0.895677733  0.283217434

 Goodness-of-fit resp. SCP Matrices
 **********************************
```

N	Av_SSQ_Diff	Correlation
1	0.000125092	0.994679763
2	0.000660168	0.972705057

Goodness-of-fit resp. Distance Matrices

****

N	Av_SSQ_Diff	Correlation
1	2.033206077	0.990027529
2	1.716943364	0.965981685

GOF=

	1
Error	0.00000
Time	0.00000
Ind.Inf.	90.000
NParms	24.000
DF	66.000
unused	.
unused	.
unused	.
unused	.
unused	.

X Configuration=

	DIM_1	DIM_2
O_01	-0.48998	-0.07507
O_02	-0.13039	0.24122
O_03	0.17143	0.57601
O_04	-0.23090	-0.54016
O_05	0.61978	0.06448
O_06	0.44304	0.18134
O_07	0.06596	-0.24954
O_08	-0.19821	-0.17442
O_09	-0.17407	0.29024
O_10	-0.07665	-0.31408

W Configuration=

	DIM_1	DIM_2
W_1	0.94028	0.23225
W_2	0.89568	0.28322

```
Eigenvalues=
   |      DIM_1      DIM_2
-------------------------
 1 |    0.95725    0.21846

Eigenvectors=
           |      DIM_1      DIM_2
---------------------------------
      O_01 |   -0.47588    0.15223
      O_02 |   -0.09539    0.31657
      O_03 |    0.23226    0.52777
      O_04 |   -0.28464   -0.46177
      O_05 |    0.59829   -0.22478
      O_06 |    0.44386   -0.01774
      O_07 |    0.03300   -0.29492
      O_08 |   -0.20974   -0.09067
      O_09 |   -0.13115    0.38902
      O_10 |   -0.11061   -0.29573
```

(b) Analysis with SUMSCAL:
We run the same analysis with the SUMSCAL method, but now for two and also one dimension:

```
optn = [ "data"      "raw" ,
         "meth"    "sumsc" ,
         "dimmax"         2 ,
         "dimmin"         1 ,
         "norm"           ];
< gof,conf,wgt,eval,evec > = mds(wxdat,optn);

                  ****************************
                  3-Way Multidimension Scaling
                  ****************************

                         Raw Input Data
   Number Rows of Matrix . . . . . .          10
   Dimension . . . . . .           2
   Amount Input Information. . . . . .        90
   Number of Parameters. . . . . .            24
   Degrees of Freedom. . . . . .              66

               *********************************
               Compute DeLEEUW-PRUZANSKY SUMSCAL
               *********************************
```

Eigenvalues of Mean Scalar Product Matrix

```
  1 :      0.9573     0.2185     0.1664    0.02963  1.191e-016
  6 :   1.633e-017 -1.537e-017 -4.755e-017 -1.052e-016 -3.128e-016
```

Unrotated common space configuration

	D_2_1	D_2_2
O_01	-0.465592568	0.071152585
O_02	-0.093331284	0.147963476
O_03	0.227242154	0.246678538
O_04	-0.278491720	-0.215828327
O_05	0.585362887	-0.105058979
O_06	0.434267720	-0.008289568
O_07	0.032286775	-0.137844258
O_08	-0.205208492	-0.042378202
O_09	-0.128315170	0.181828134
O_10	-0.108220302	-0.138223400

Undiagonalized Weight Matrices:
***

Weight matrix: 1

	D_2_1	D_2_2
D_2_1	1.017745116	
D_2_2	-0.027635222	0.907661656

Weight matrix: 2

	D_2_1	D_2_2
D_2_1	0.982254884	
D_2_2	0.027635222	1.092338344

Iterative Simultaneous Diagonalization:
***

Iter	ssq-off-dia	max(ang)	min(cos)
1	0.055270443	1.3408e+154	0.000000000
2	4.3715e-016	0.232654169	0.973057875
3	4.3715e-016	2.8166e-017	1.000000000

```
Residuals SIMDIAG: nit = 2
--------------------------

  1 :   4.627e-032  4.93e-032

                  Orthogonal Rotation Matrix

                              D_2_1        D_2_2
                D_2_1   0.973057875 -0.230560993
                D_2_2   0.230560993  0.973057875

               Rotated common space configuration

                            D_2_1        D_2_2
                O_01  -0.469453525 -0.038111902
                O_02  -0.124931347  0.122458472
                O_03   0.164245319  0.292425671
                O_04  -0.221226968 -0.274222781
                O_05   0.593814469  0.032733382
                O_06   0.424478876  0.092058967
                O_07   0.063198410 -0.126686370
                O_08  -0.189908979 -0.088549516
                O_09  -0.166780562  0.147344825
                O_10  -0.073435693 -0.159450748

                 Diagonalized Weight Matrices:
                 *****************************

                      Weight matrix:    1

                              D_2_1        D_2_2
                D_2_1   1.024293137
                D_2_2  -2.1511e-016  0.901113634

                      Weight matrix:    2

                              D_2_1        D_2_2
                D_2_1   0.975706863
                D_2_2  -2.2204e-016  1.098886366

               Schoenemann's Testindizes for SUMSCAL

                     1  0.964379639  1.5832e-016
                     2  0.885697342  1.5163e-016
```

```
******************************
Result Normalized Configuration
******************************

   Common Space Configuration

            D_2_1        D_2_2
O_01  -0.489977791 -0.075071833
O_02  -0.130393281  0.241215514
O_03   0.171426039  0.576012484
O_04  -0.230898897 -0.540156903
O_05   0.619775732  0.064477364
O_06   0.443036873  0.181335360
O_07   0.065961412 -0.249543518
O_08  -0.198211702 -0.174422535
O_09  -0.174072122  0.290236005
O_10  -0.076646263 -0.314081938

   Subjective Weight Matrices

            D_2_1        D_2_2
W_1   0.940278879  0.232245207
W_2   0.895677733  0.283217434


  Goodness-of-fit resp. SCP Matrices
  **********************************

       N  Av_SSQ_Diff  Correlation
       1  0.000125092  0.994679763
       2  0.000660168  0.972705057

 Goodness-of-fit resp. Distance Matrices
 ***************************************

       N  Av_SSQ_Diff  Correlation
       1  2.033206077  0.990027529
       2  1.716943364  0.965981685
```

It follows the analysis in one dimension:

```
*************************************************
Compute DeLeeuw-Pruzansky SUMSCAL for 1 Dimension(s)
*************************************************

   Unrotated common space configuration
```

```
                D_1_1
O_01  -0.465592568
O_02  -0.093331284
O_03   0.227242154
O_04  -0.278491720
O_05   0.585362887
O_06   0.434267720
O_07   0.032286775
O_08  -0.205208492
O_09  -0.128315170
O_10  -0.108220302

Undiagonalized Weight Matrices:
*******************************

      Weight matrix:    1

                D_1_1
D_1_1   1.017745116

      Weight matrix:    2

                D_1_1
D_1_1   0.982254884

Schoenemann's Testindizes for SUMSCAL

      1  0.908403857  0.000000000
      2  0.816123639  0.000000000

*******************************
Result Normalized Configuration
*******************************

   Common Space Configuration

                D_1_1
O_01  -0.475875172
O_02  -0.095392504
O_03   0.232260793
O_04  -0.284642206
O_05   0.598290617
O_06   0.443858516
O_07   0.032999828
```

```
O_08  -0.209740518
O_09  -0.131149009
O_10  -0.110610345

Subjective Weight Matrices

            D_1_1
W_1   0.974237878
W_2   0.940264806

Goodness-of-fit resp. SCP Matrices
**********************************

     N  Av_SSQ_Diff  Correlation
     1  0.000574718  0.975346430
     2  0.001385490  0.941856456

Goodness-of-fit resp. Distance Matrices
***************************************

     N  Av_SSQ_Diff  Correlation
     1  0.000000000  1.000000000
     2  0.000000000  1.000000000
```

(c) Analysis with INDSCAL:

```
optn = [ "data"      "raw" ,
         "meth"    "indsc" ,
         "ndim"          2 ,
         "norm"           ];
< gof,conf,wgt,eval,evec > = mds(wxdat,optn);
```

The following output of the iteration history could be restricted to larger steps using the `plin` option (default is 10):

```
Iterative Canonical Decomposition:
**********************************

Iter  Way    Max. Dev.     Maximum     Residual
   1    1  0.092338344
   1    2  0.002573734  0.092338344  0.275733651
  11    1  0.000126273
  11    2  0.001420932  0.001420932  0.275619458
  21    1  9.1720e-005
  21    2  0.001341886  0.001341886  0.275430703
  31    1  5.9351e-005
```

```
 31    2  0.001262113  0.001262113  0.275255123
 41    1  3.3093e-005
 41    2  0.001178954  0.001178954  0.275097042
  ..........................................
631    1  3.3370e-006
631    2  1.2614e-005  1.2614e-005  0.274047651
641    1  3.1349e-006
641    2  1.1837e-005  1.1837e-005  0.274047522
651    1  2.9452e-006
651    2  1.1108e-005  1.1108e-005  0.274047402
661    1  2.7671e-006
661    2  1.0426e-005  1.0426e-005  0.274047292
664    0  9.9616e-006  9.9616e-006  0.274047235

Residuals CADECO: nit = 664
---------------------------

  1 :    0.006999    0.03625

            Result non-normalized configuration X:
            ***************************************

                 Common Space Configuration

                      DIM_1        DIM_2
               O_01  -0.469705827 -0.036553819
               O_02  -0.124329431  0.121437948
               O_03   0.164347235  0.294394706
               O_04  -0.222344182 -0.271971552
               O_05   0.594691055  0.028653940
               O_06   0.423981427  0.094048475
               O_07   0.063142821 -0.127689246
               O_08  -0.190385558 -0.087104099
               O_09  -0.165710781  0.144643058
               O_10  -0.073686760 -0.159859411

                 Common Space Configuration

                      DIM_1        DIM_2
               O_01  -0.469705827 -0.036553819
               O_02  -0.124329431  0.121437948
               O_03   0.164347235  0.294394706
               O_04  -0.222344182 -0.271971552
               O_05   0.594691055  0.028653940
               O_06   0.423981427  0.094048475
```

```
O_07   0.063142821 -0.127689246
O_08  -0.190385558 -0.087104099
O_09  -0.165710781  0.144643058
O_10  -0.073686760 -0.159859411

   Subjective Weight Matrices

            DIM_1        DIM_2
 W_1   1.023979211  0.899153581
 W_2   0.975339995  1.102091165

Result Normalized Configuration
*******************************

   Common Space Configuration

            DIM_1        DIM_2
O_01  -0.489950010 -0.072139585
O_02  -0.129687993  0.239659861
O_03   0.171430553  0.580992970
O_04  -0.231927151 -0.536740493
O_05   0.620322065  0.056549038
O_06   0.442254902  0.185606268
O_07   0.065864258 -0.251996903
O_08  -0.198591120 -0.171901425
O_09  -0.172852867  0.285455541
O_10  -0.076862638 -0.315485272

   Common Space Configuration

            DIM_1        DIM_2
O_01  -0.489950010 -0.072139585
O_02  -0.129687993  0.239659861
O_03   0.171430553  0.580992970
O_04  -0.231927151 -0.536740493
O_05   0.620322065  0.056549038
O_06   0.442254902  0.185606268
O_07   0.065864258 -0.251996903
O_08  -0.198591120 -0.171901425
O_09  -0.172852867  0.285455541
O_10  -0.076862638 -0.315485272

   Subjective Weight Matrices

            DIM_1        DIM_2
```

```
      W_1   0.941108062  0.230861820
      W_2   0.896405242  0.282966978

   Goodness-of-fit resp. SCP Matrices
   **********************************

        N  Av_SSQ_Diff  Correlation
        1  0.000127246  0.994584218
        2  0.000659062  0.972744353

  Goodness-of-fit resp. Distance Matrices
  ***************************************

        N  Av_SSQ_Diff  Correlation
        1  2.075138105  0.989819063
        2  1.718323923  0.965938432
```

2. Scalar product and distance data in tensor form `D[2,n,n]`:
   The first part of randomly generating the `coef` and `wgt` data is the same as for the raw data tensor `wxdat` above.
   The tensor data `eudis` of distances are then obtained using the `dist` function:

```
/* [6] Weighted Euclidean Distances: WEUCDIS() */
real eudis[p,n,n];
for (j = 1; j <= p; j++) {
  wv = wght[j,];
  xcof = coef .* (none * wv);
  optn = [ 0, 0 ];
  eudis[j,,] = dist(xcof,"L2",optn);
  /* print "Eudis[j,,]=", eudis[j,,]; */
}
print "Tensor EUDIS=",eudis;
```

```
Tensor EUDIS=
                  ****************************
                  Tensor eudis with 3 Dimensions
                  ****************************


                            eudis_1
                            *******

                 Dense Symmetric Matrix (10 by 10)
```

S	1	2	3	4	5
1	0.0000000				
2	12.517277	0.0000000			
3	22.152531	9.8394964	0.0000000		
4	12.356763	12.940254	20.551525	0.0000000	
5	33.567743	22.870327	15.553271	28.148374	0.0000000
6	28.987725	17.472745	10.667907	22.715945	8.6757329
7	17.231406	9.4344830	12.034581	10.724700	17.628005
8	9.2229700	7.0525519	15.536413	6.9026711	25.157692
9	15.117752	6.4949865	12.174883	13.858909	25.699440
10	15.344827	9.7586323	15.262637	6.3637807	23.108129

S	6	7	8	9	10
6	0.0000000				
7	13.096467	0.0000000			
8	20.315094	8.0987128	0.0000000		
9	18.592868	12.836619	10.690362	0.0000000	
10	16.807093	6.6596174	7.0984902	10.001059	0.0000000

eudis_2
****

Dense Symmetric Matrix (10 by 10)

S	1	2	3	4	5
1	0.0000000				
2	5.9135599	0.0000000			
3	10.566485	5.3508700	0.0000000		
4	6.6365373	6.2318712	10.132854	0.0000000	
5	14.788985	10.194800	7.9197796	12.671486	0.0000000
6	13.704587	8.4382427	5.2512397	10.573009	5.8389505
7	7.9080518	4.4822392	6.8526023	4.9812874	7.8457810
8	4.3468595	3.3647800	7.6585561	3.8384410	11.074632
9	8.6863458	4.5453515	7.4888400	6.8377826	12.298703
10	8.2577739	5.2962197	8.4694364	3.0972848	10.840219

S	6	7	8	9	10
6	0.0000000				
7	7.0806884	0.0000000			

```
   8 |   9.6791208  3.7312506  0.0000000
   9 |   9.1360795  6.8728797  6.4520669  0.0000000
  10 |   8.3133128  3.9235426  4.6062928  4.8329621  0.0000000
```

All the results obtained with COSPA, SUMSCAL, and INDSCAL are the same as those obtained with the raw data input. Here we only show the `mds` call for COSPA:

```
optn = [ "data"      "dis" ,
         "meth"    "cospa" ,
         "nobs"         10 ,
         "ndim"          2 ,
         "norm"           ];
< gof,conf,wgt,eval,evec > = mds(eudis,optn);
print "GOF=",gof;
print "X Configuration=",conf;
print "W Configuration=",wgt;
print "Eigenvalues=",eval;
print "Eigenvectors=",evec;
```

The tensor data `scalp` of scalar products are then obtained using the simple code:

```
/* [6] Weighted Configuration: Y = X * W */
real scalp[p,n,n];
for (j = 1; j <= p; j++) {
  wv = wght[j,];
  xdat = coef .* (none * wv);
  /* print "Xdat=", xdat; */
  scalp[j,,] = xdat * xdat';
}
print "Tensor SCALP=",scalp;
```

```
Tensor SCALP=
                  ****************************
                  Tensor scalp with 3 Dimensions
                  ****************************


                            scalp_1
                            *******

                Dense Symmetric Matrix (10 by 10)

     S |          1          2          3          4          5
```

```
 ------------------------------------------------------------
 1 |   239.35651
 2 |   56.629284  30.584276
 3 |  -80.019400  12.553964  91.339342
 4 |   104.65673 -7.1096775 -104.18964  122.64655
 5 |  -259.82674 -62.342084  108.60925 -150.95050  367.78340
 6 |  -208.29347 -45.183882  80.939938 -104.51143  238.42991
 7 |  -19.625132 -20.055300 -17.588603  12.970976  37.675707
 8 |   101.15092  14.427146 -51.016135  61.504092 -108.55878
 9 |   44.237560  33.032238  10.388312  4.1211227 -107.50639
10 |   21.733749 -12.535965 -51.017027  60.861770 -63.313758


 S |           6          7          8          9          10
 ------------------------------------------------------------
 6 |   184.34477
 7 |   15.570962  18.314592
 8 |  -90.174890  0.3669736  48.008505
 9 |  -41.842453 -34.399568  5.6948546  77.665048
10 |  -29.279458  6.7693928  18.597319  8.6092796  39.574698
```

scalp_2
****

Dense Symmetric Matrix (10 by 10)

```
 S |           1          2          3          4          5
 ------------------------------------------------------------
 1 |   53.368298
 2 |   12.654620  6.9111339
 3 |  -15.236448  3.0443698  27.809416
 4 |   17.923227 -2.7016504 -24.171767  26.521784
 5 |  -45.452089 -11.290610  19.764048 -29.801599  74.441589
 6 |  -45.209182 -10.131875  22.131477 -20.618844  42.188651
 7 |  -1.8816273 -3.8868014 -6.8715051  3.5571455  9.1455204
 8 |   22.642510  3.2006496 -10.016078  11.300032 -18.696990
 9 |   1.8234916  5.9911013 -1.2710098  2.7489006 -25.542608
10 |  -0.6327998 -3.7909387 -15.182503  15.242772 -14.755914


 S |           6          7          8          9          10
 ------------------------------------------------------------
 6 |   44.029056
 7 |  -0.3506802  5.4057317
 8 |  -19.422208  1.1477050  10.811910
 9 |  -6.8538026 -8.0497273 -2.5429844  25.731289
```

```
  10 |  -5.7625912  1.7842387  1.5754542  7.9653492  13.556932
```

Running the following call of `mds` will generate the same results as reported above for other data input:

```
optn = [ "data"      "scp" ,
         "meth"    "cospa" ,
         "nobs"         10 ,
         "ndim"          2 ,
         "norm"           ];
< gof,conf,wgt,eval,evec > = mds(scalp,optn);
```

3. Test Example from the KYST Manual: Unsymmetric Matrix of Similarity Measures: Regression: Descending

   First we use descending isotone (monotone) regression for the relationship among data and model values:

```
simi = [ 9.30  0.57  6.10  2.00  1.50 ,
         1.00  8.50  5.50  3.30  4.00 ,
         2.10  5.30  7.30  3.90  1.10 ,
         5.00  6.20  1.60  8.90  2.80 ,
         1.30  4.10  1.00  2.50  8.80 ];


print "KYST MDS without initial values: Descending";
optn = [ "data"       "simi" ,
         "unsym"              ,
         "meth"       "kyst" ,
         "regres"     "desc" ,
         "pre-it"          3 ,
         "coord"     "rotat" ,
         "dimmax"          3 ,
         "dimmin"          1 ,
         "phis"              ,
         "norm"             ];
< gof,conf > = mds(simi,optn);


              *********************************
              NonMetric Multidimensional Scaling
              *********************************

            Similarity Input Data (Unsymmetric)
 Number Rows of Matrix . . . . . .          5
 Maximum Dimension . . . . . .              3
 Minimum Dimension . . . . . .              1
```

```
   Amount Input Information. . . . . .          25
   Number of Parameters. . . . . .              15
   Degrees of Freedom. . . . . .                10
   Stress Formula. . . . . .                     1
   Diagonal  . . . . . .                  Included
   Isotone Regression  . . . . . .      Descending
   Tie Treatment . . . . . .               Primary
   Maximum Number Iterations . . . . . .      5000
   TORSCA Preiterations. . . . . .               3


 Similarity Matrix: 1
 --------------------

   1|        9.3       0.57        6.1          2        1.5
   2|          1        8.5        5.5        3.3          4
   3|        2.1        5.3        7.3        3.9        1.1
   4|          5        6.2        1.6        8.9        2.8
   5|        1.3        4.1          1        2.5        8.8


          ******************************************
          Compute Kruskal-Young-Shepard-Torgerson KYST
          ******************************************
         There are 25 data values, split into 1 lists.

Torgerson-Young Quasi-Nonmetric Procedure for Initial Configuration

                     Iteration      Stress

                             0  0.022638560
                             1  0.002207875
                             2  0.000459813
                             3  0.000153818
 [note] file tmds1.inp, line 63: Maximum number of pre-iterations 3
           reached.
The best initial configuration by Torgerson-Young quasi-nonmetric
method has Stress=0.000153818 Formula 1


             History of Iteration for 3 Dimensions
             *************************************

   Iter STRESS   SRAT SRATAV CAGRCL  COSAV  ACSAV    SFGR    STEP

      0  0.010  0.800  0.800  0.000  0.000  0.000  0.0010  0.0002
        Satisfactory Stress was reached after 0 iterations.
```

Final Configuration (Stress= 0.00989601 Formula 1)

	D_3_1	D_3_2	D_3_3
O_1	1.075123487	-0.513110454	-0.160950519
O_2	-0.696452481	0.652971078	-0.001235367
O_3	0.568985138	0.719528505	-0.156694064
O_4	0.008885202	-0.233080220	0.586641998
O_5	-0.956541346	-0.626308909	-0.267762048

Group	Count	Stress	Regression
1	25	0.00989601	Descending

History of Iteration for 2 Dimensions

Iter	STRESS	SRAT	SRATAV	CAGRCL	COSAV	ACSAV	SFGR	STEP
0	0.059	0.800	0.800	0.000	0.000	0.000	0.0039	0.0057
1	0.057	0.968	0.852	0.999	0.659	0.659	0.0037	0.0155
2	0.052	0.916	0.873	0.988	0.876	0.876	0.0031	0.0530
3	0.041	0.785	0.843	0.357	0.533	0.533	0.0014	0.1104
	0.060			-0.658				0.0110
4	0.039	0.966	0.882	0.757	0.681	0.681	0.0010	0.0291
5	0.037	0.948	0.903	0.216	0.374	0.374	0.0005	0.0476
	0.051			-0.940				0.0048
6	0.037	0.996	0.933	-0.490	-0.196	0.451	0.0002	0.0027
7	0.037	0.999	0.955	-0.283	-0.253	0.340	0.0002	0.0016
8	0.037	0.999	0.969	0.007	-0.081	0.120	0.0001	0.0012
9	0.037	1.000	0.979	0.219	0.117	0.185	0.0001	0.0011
10	0.037	1.000	0.986	-0.272	-0.140	0.242	0.0001	0.0007
11	0.037	1.000	0.991	-0.381	-0.299	0.334	0.0000	0.0003
12	0.037	1.000	0.994	0.111	-0.028	0.187	0.0000	0.0002
13	0.037	1.000	0.996	0.415	0.264	0.337	0.0000	0.0003
14	0.037	1.000	0.997	0.008	0.095	0.120	0.0000	0.0002
15	0.037	1.000	0.998	-0.704	-0.432	0.505	0.0000	0.0001
16	0.037	1.000	0.999	0.850	0.414	0.733	0.0000	0.0001
17	0.037	1.000	0.999	-0.496	-0.187	0.577	0.0000	0.0000

Minimum was achieved after 17 iterations.

Final Configuration (Stress= 0.0371116 Formula 1)

	D_2_1	D_2_2
O_1	1.147921385	-0.496658734
O_2	-0.646499738	0.672365556
O_3	0.649763151	0.672655794
O_4	-0.001631430	-0.354985561
O_5	-1.149553368	-0.493377055

Group	Count	Stress	Regression
1	25	0.03711159	Descending

History of Iteration for 1 Dimensions

****

Iter	STRESS	SRAT	SRATAV	CAGRCL	COSAV	ACSAV	SFGR	STEP
0	0.212	0.800	0.800	0.000	0.000	0.000	0.0109	0.0575
1	0.204	0.963	0.851	0.850	0.561	0.561	0.0006	0.1358
2	0.231	1.134	0.937	-0.946	-0.434	0.815	0.0187	0.0501
3	0.215	0.928	0.934	0.989	0.505	0.930	0.0110	0.0639
4	0.205	0.956	0.941	0.501	0.502	0.647	0.0029	0.1008
	0.213			-0.979				0.0101
5	0.205	0.997	0.959	0.997	0.829	0.878	0.0022	0.0267
6	0.204	0.997	0.972	-0.903	-0.314	0.895	0.0008	0.0094
	0.204			-0.977				0.0009
7	0.204	1.000	0.981	1.000	0.553	0.964	0.0006	0.0012
8	0.204	1.000	0.987	0.999	0.847	0.987	0.0004	0.0028
9	0.204	1.000	0.991	-0.649	-0.140	0.764	0.0001	0.0012
	0.204			-1.000				0.0001
10	0.204	1.000	0.994	1.000	0.612	0.920	0.0000	0.0002
11	0.204	1.000	0.996	1.000	0.868	0.973	0.0000	0.0004
	0.204			-1.000				0.0000
12	0.204	1.000	0.997	1.000	0.955	0.991	0.0000	0.0001
	0.204			-1.000				0.0000
13	0.204	1.000	0.998	1.000	0.985	0.997	0.0000	0.0000
	0.204			-0.990				0.0000
14	0.204	1.000	0.999	1.000	0.995	0.999	0.0000	0.0000
15	0.204	1.000	0.999	1.000	0.998	1.000	0.0000	0.0000

Minimum was achieved after 15 iterations.

Final Configuration (Stress= 0.203946 Formula 1)

	D_1_1
O_1	1.394282445

```
                    O_2  -0.663840798
                    O_3   0.746266448
                    O_4  -0.042636844
                    O_5  -1.434071249


              Group Count      Stress Regression

                  1    25  0.20394609 Descending
```

4. Test Example from the KYST Manual: Unsymmetric Matrix of Similarity Measures: Regression: Third Order Polynomial

```
simi = [ 9.30  0.57  6.10  2.00  1.50 ,
         1.00  8.50  5.50  3.30  4.00 ,
         2.10  5.30  7.30  3.90  1.10 ,
         5.00  6.20  1.60  8.90  2.80 ,
         1.30  4.10  1.00  2.50  8.80 ];


print "KYST MDS without initial values: Polynomial Degree=3";
optn = [ "data"       "simi" ,
         "unsym"             ,
         "meth"       "kyst" ,
         "regres"     "poly" ,
         "poldeg"          3 ,
         "pre-it"          3 ,
         "coord"     "rotat" ,
         "dimmax"          3 ,
         "dimmin"          1 ,
         "phis"              ,
         "norm"             ];
< gof,conf > = mds(simi,optn);


              ******************************
              Metric Multidimensional Scaling
              ******************************

            Similarity Input Data (Unsymmetric)
 Number Rows of Matrix . . . . . .          5
 Maximum Dimension . . . . . .              3
 Minimum Dimension . . . . . .              1
 Amount Input Information. . . . . .       25
 Number of Parameters. . . . . .           15
 Degrees of Freedom. . . . . .             10
 Stress Formula. . . . . .                  1
```

```
Diagonal  . . . . . .  Included
Polynomial Regression Degree. . . . . .  3
Maximum Number Iterations . . . . . .  5000
TORSCA Preiterations. . . . . .  3

        ******************************************
        Compute Kruskal-Young-Shepard-Torgerson KYST
        ******************************************
       There are 25 data values, split into 1 lists.

Torgerson-Young Quasi-Nonmetric Procedure for Initial Configuration

                    Iteration      Stress

                            0  0.022638560
                            1  0.002207875
                            2  0.000459813
                            3  0.000153818
 [note] file tmds1.inp, line 42: Maximum number of pre-iterations 3
          reached.
The best initial configuration by Torgerson-Young quasi-nonmetric
method has Stress=0.000153818 Formula 1

            History of Iteration for 3 Dimensions
            *************************************

  Iter STRESS   SRAT SRATAV CAGRCL  COSAV  ACSAV    SFGR    STEP

    0  0.169  0.800  0.800  0.000  0.000  0.000  0.0069  0.0293
    1  0.163  0.968  0.852  0.974  0.643  0.643  0.0056  0.0776
    2  0.153  0.939  0.880  0.738  0.706  0.706  0.0034  0.2093
    3  0.155  1.008  0.921 -0.703 -0.224  0.704  0.0057  0.0997
    4  0.147  0.948  0.930 -0.463 -0.382  0.545  0.0022  0.0464
    5  0.146  0.998  0.952 -0.837 -0.682  0.738  0.0021  0.0153
    6  0.146  0.995  0.966  0.934  0.384  0.867  0.0006  0.0152
    7  0.146  1.000  0.977 -0.897 -0.462  0.887  0.0008  0.0048
    8  0.146  0.999  0.985  0.983  0.492  0.950  0.0003  0.0054
    9  0.146  1.000  0.990 -0.819 -0.374  0.864  0.0002  0.0018
   10  0.146  1.000  0.993  0.401  0.138  0.559  0.0000  0.0012
   11  0.146  1.000  0.995  0.574  0.426  0.569  0.0000  0.0016
   12  0.146  1.000  0.997 -0.448 -0.151  0.489  0.0001  0.0008
   13  0.146  1.000  0.998 -0.553 -0.416  0.531  0.0000  0.0003
   14  0.146  1.000  0.999 -0.631 -0.558  0.597  0.0000  0.0001
   15  0.146  1.000  0.999  0.901  0.405  0.798  0.0000  0.0001
          Minimum was achieved after 15 iterations.
```

```
Final Configuration (Stress= 0.145597 Formula 1)

             D_3_1        D_3_2        D_3_3
O_1   0.966987925 -0.512572882 -0.077858169
O_2  -0.609811159  0.628542479  0.045316107
O_3   0.480826435  0.706684142 -0.399330276
O_4  -0.075197068 -0.128523217  0.908024094
O_5  -0.762806134 -0.694130522 -0.476151756
```

The regression coefficients start with the intercept and it is of no surprise that the coefficient of the order 3 term is negative indicating an descending behavior:

```
Group Count      Stress Regression Coefficients

    1    25  0.14559717   2.003  -0.206   0.030  -0.004


          History of Iteration for 2 Dimensions
          *************************************

Iter STRESS   SRAT SRATAV CAGRCL  COSAV  ACSAV    SFGR    STEP

   0  0.209  0.800  0.800  0.000  0.000  0.000  0.0054  0.0281
   1  0.206  0.986  0.858  0.929  0.613  0.613  0.0036  0.0714
   2  0.203  0.983  0.898  0.149  0.307  0.307  0.0028  0.1094
   3  0.205  1.012  0.934 -0.635 -0.315  0.524  0.0073  0.0547
   4  0.200  0.973  0.947  0.532  0.244  0.529  0.0015  0.0535
   5  0.200  1.002  0.965 -0.631 -0.333  0.596  0.0035  0.0232
   6  0.199  0.994  0.975  0.775  0.398  0.714  0.0010  0.0260
   7  0.199  1.000  0.983 -0.610 -0.267  0.645  0.0014  0.0109
   8  0.199  0.999  0.988  0.742  0.399  0.709  0.0005  0.0119
   9  0.199  1.000  0.992 -0.035  0.113  0.264  0.0005  0.0098
  10  0.199  1.000  0.995 -0.483 -0.281  0.409  0.0005  0.0048
  11  0.198  1.000  0.996  0.293  0.098  0.332  0.0002  0.0036
  12  0.198  1.000  0.998  0.805  0.564  0.644  0.0002  0.0059
  13  0.198  1.000  0.998  0.068  0.237  0.264  0.0002  0.0064
  14  0.198  1.000  0.999 -0.851 -0.481  0.652  0.0005  0.0022
  15  0.198  1.000  0.999  0.985  0.486  0.872  0.0003  0.0025
        Minimum was achieved after 15 iterations.


Final Configuration (Stress= 0.198448 Formula 1)

                      D_2_1        D_2_2
```

```
O_1   1.144281569 -0.497908299
O_2  -0.620555127  0.658039311
O_3   0.572111824  0.801728943
O_4  -0.004133515 -0.487920895
O_5  -1.091704751 -0.473939059

Group Count      Stress Regression Coefficients

    1    25  0.19844843   2.754  -0.894   0.176  -0.012

          History of Iteration for 1 Dimensions
          *************************************

Iter STRESS   SRAT SRATAV CAGRCL  COSAV  ACSAV    SFGR    STEP

   0  0.332  0.800  0.800  0.000  0.000  0.000  0.0078  0.0647
      0.330               -0.967                        0.0065
   1  0.332  0.998  0.861  1.000  0.660  0.660  0.0067  0.0175
   2  0.330  0.996  0.904  0.998  0.883  0.883  0.0037  0.0593
      0.331               -0.989                        0.0059
   3  0.330  0.999  0.935  1.000  0.960  0.960  0.0027  0.0210
   4  0.330  0.999  0.956 -0.929 -0.287  0.939  0.0009  0.0076
      0.330               -0.998                        0.0008
   5  0.330  1.000  0.970  1.000  0.563  0.979  0.0007  0.0010
   6  0.330  1.000  0.980  1.000  0.851  0.993  0.0005  0.0024
   7  0.330  1.000  0.987  0.995  0.946  0.994  0.0001  0.0071
      0.330               -0.999                        0.0007
   8  0.330  1.000  0.991 -0.430  0.038  0.622  0.0000  0.0004
      0.330               -0.992                        0.0000
   9  0.330  1.000  0.994  0.466  0.321  0.519  0.0000  0.0000
      0.330               -0.995                        0.0000
  10  0.330  1.000  0.996  0.953  0.738  0.805  0.0000  0.0000
      0.330               -1.000                        0.0000
  11  0.330  1.000  0.997  0.998  0.910  0.933  0.0000  0.0000
      0.330               -1.000                        0.0000
  12  0.330  1.000  0.998  1.000  0.969  0.977  0.0000  0.0000
      0.330               -1.000                        0.0000
  13  0.330  1.000  0.999  1.000  0.990  0.992  0.0000  0.0000
      0.330               -1.000                        0.0000
  14  0.330  1.000  0.999  1.000  0.996  0.997  0.0000  0.0000
        Minimum was achieved after 14 iterations.

      Final Configuration (Stress= 0.329676 Formula 1)
```

	D_1_1
O_1	1.452973688
O_2	-0.710018727
O_3	0.707880939
O_4	-0.080754068
O_5	-1.370081832

Group	Count	Stress	Regression Coefficients			
1	25	0.32967640	3.139	-1.212	0.223	-0.014

3.5 Function `setdiff`

< c,indx > = setdiff(a,b)

Purpose: There are two different input versions:

1. Arguments `a` and `b` are scalars or vectors:
2. Arguments `a` and `b` are matrices with the same column number:

After reducing `a` and `b` to sets of unique entries (removing duplicates) this function returns in `c` the set difference of `a` with `b`.

Input: **a** can be scalar, vector, or matrix

b can be scalar, vector, or matrix

Output: **c** Returns the K unique entries which are in `a` **but not** in `b` (without replications). The entries of `c` are sorted with numeric values at first and string values last. If $K = 0$ a missing value is returned.

indx Returns the $K \times 2$ matrix indicating which of the entries of `c` is from `a` (first column) and which is from `b` (second column). For this function the second column contains only values -1 indicating that none of the entries in `c` could come from `b`.

Restrictions: 1. The input arguments can be numeric, including missing values, or string or of mixed data type.

2. The arguments `a`, and `b` can be scalar, vector, or matrix. If `a` or `b` is matrix, the other must be a matrix with the same number of columns.

Relationships: setisect(), setmembr(), setunion(), setxor(), unique()

Examples: 1. Numerical Data:

```
a1 = [ 2#3 3#4 2 3 1 4 5 ];
a2 = [ 2#3. 3#4. . . . 2. 3. 1 4. 6. . . . ];
< c1,ind > = setdiff(a1,a2);
print "C1=",c1;
print "Ind=",ind;
```

```
C1=
   |         1
---------------
 1 |    5.0000
```

```
Ind=
   |    1    2
--------------
 1 |   10   -1
```

2. String Data:

```
a3 = [ "one" "two" "three" "four" "five" "one" "three" "two" ];
b3 = [ "six" "seven" "one" ];
< c2,ind > = setdiff(a3,b3);
print "C2=",c2;
print "Ind=",ind;

< c3,ind > = setdiff(b3,a3);
print "C3=",c3;
print "Ind=",ind;
```

```
C2=
   |       1       2       3       4
-------------------------------------
 1 |    five    four   three     two


Ind=
   |   1   2
------------
 1 |   5  -1
 2 |   4  -1
 3 |   3  -1
 4 |   2  -1


C3=                                        Ind=
   |       1       2                          |   1   2
--------------------                       ------------
 1 |   seven     six                         1 |   2  -1
                                             2 |   1  -1
```

3.6 Function `setisect`

< c,indx > = setisect(a,b)

Purpose: There are two different input versions:

1. Arguments `a` and `b` are scalars or vectors:
2. Arguments `a` and `b` are matrices with the same column number:

After reducing `a` and `b` to sets of unique entries (removing duplicates) this function returns in `c` the set intersection of `a` with `b`.

Input: **a** can be scalar, vector, or matrix

b can be scalar, vector, or matrix

Output: **c** Returns the K unique entries which are in `a` **and** also in `b` (without replications). The entries of `c` are sorted with numeric values at first and string values last. If $K = 0$ a missing value is returned.

indx Returns the $K \times 2$ matrix indicating which of the entries of `c` is from `a` (first column) and which is from `b` (second column). For this function both columns cannot contain negative values.

Restrictions: 1. The input arguments can be numeric, including missing values, or string or of mixed data type.

2. The arguments `a`, and `b` can be scalar, vector, or matrix. If `a` or `b` is matrix, the other must be a matrix with the same number of columns.

Relationships: setdiff(), setmembr(), setunion(), setxor(), unique()

Examples: 1. Numerical Data:

```
a1 = [ 2#3 3#4 2 3 1 4 5 ];
a2 = [ 2#3. 3#4. . . . 2. 3. 1 4. 6. . . . ];
< c1,ind > = setisect(a1,a2);
print "C1=",c1;
print "Ind=",ind;
```

```
C1=
   |         1         2         3         4
---------------------------------------------
 1 |   1.00000    2.0000    3.0000    4.0000
```

```
Ind=
   |    1    2
--------------
 1 |    8   11
 2 |    6    9
 3 |    2   10
 4 |    4    3
```

2. String Data:

```
a3 = [ "one" "two" "three" "four" "five" "one" "three" "two" ];
b3 = [ "six" "seven" "one" ];
< c2,ind > = setisect(a3,b3);
print "C2=",c2;
print "Ind=",ind;
```

```
C2=
   |     1
----------
 1 |   one

Ind=
   |   1   2
------------
 1 |   1   3
```

```
< c3,ind > = setisect(b3,a3);
print "C3=",c3;
print "Ind=",ind;
```

```
C3=
   |     1
----------
 1 |   one

Ind=
   |   1   2
------------
 1 |   3   1
```

3.7 Function `setmembr`

c = setmembr(a,b)

Purpose: There are two different input versions:

1. Arguments `a` and `b` are scalars or vectors:
2. Arguments `a` and `b` are matrices with the same column number:

Input: **a** can be scalar, vector, or matrix

b can be scalar, vector, or matrix

Output: This function returns a binary object `c` of the same dimension as argument `a` in which entries of one indicate that the entry of `a` is entry of `b`. A zero indicates that the corresponding entry of `a` is not in `b`.

Restrictions: 1. The input arguments can be numeric, including missing values, or string or of mixed data type.

2. The arguments `a`, and `b` can be scalar, vector, or matrix. If `a` or `b` is matrix, the other must be a matrix with the same number of columns.

Relationships: setdiff(), setisect(), setunion(), setxor(), unique()

Examples: 1. Numerical Data:

```
a1 = [ 2#3 3#4 2 3 1 4 5 ];
a2 = [ 2#3. 3#4. . . . 2. 3. 1 4. 6. . . . ];
c1 = setmembr(a1,a2);
print "C1=",c1;
```

```
C1=
   |   1   2   3   4   5   6   7   8   9  10
-------------------------------------------
 1 |   1   1   1   1   1   1   1   1   1   0
```

2. String Data:

```
a3 = [ "one" "two" "three" "four" "five" "one" "three" "two" ];
b3 = [ "six" "seven" "one" ];
c2 = setmembr(a3,b3);
print "C2=",c2;
```

```
C2=
   |   1   2   3   4   5   6   7   8
-----------------------------------
 1 |   1   0   0   0   0   1   0   0
```

```
   c3 = setmembr(b3,a3);
   print "C3=",c3;
```

```
C3=
   |   1   2   3
---------------
 1 |   0   0   1
```

3.8 Function `setunion`

< c,indx > = setunion(a,b)

Purpose: There are two different input versions:

1. Arguments `a` and `b` are scalars or vectors:
2. Arguments `a` and `b` are matrices with the same column number:

After reducing `a` and `b` to sets of unique entries (removing duplicates) this function returns in `c` the set union (inclusive or) of `a` with `b`.

Input: **a** can be scalar, vector, or matrix

b can be scalar, vector, or matrix

Output: **c** Returns the K unique entries which are in `a or` in `b` (without replications). The entries of `c` are sorted with numeric values at first and string values last. If $K = 0$ a missing value is returned.

indx Returns the $K \times 2$ matrix indicating which of the entries of `c` is from `a` (first column) and which is from `b` (second column).

Restrictions: 1. The input arguments can be numeric, including missing values, or string or of mixed data type.

2. The arguments `a`, and `b` can be scalar, vector, or matrix. If `a` or `b` is matrix, the other must be a matrix with the same number of columns.

Relationships: setdiff(), setisect(), setmembr(), setxor(), unique()

Examples: 1. Numerical Data:

```
a1 = [ 2#3 3#4 2 3 1 4 5 ];
a2 = [ 2#3. 3#4. . . . 2. 3. 1 4. 6. . . . ];
< c1,ind > = setunion(a1,a2);
print "C1=",c1;
print "Ind=",ind;
```

```
C1=
   |         1         2         3         4
---------------------------------------------
 1 |    .        1.00000    2.0000    3.0000

   |         5         6         7
-----------------------------------
 1 |    4.0000    5.0000    6.0000
```

```
Ind=
   |    1    2
--------------
 1 |   -1    6
 2 |    8   11
 3 |    6    9
 4 |    2   10
 5 |    4    3
 6 |   10   -1
 7 |   -1   13
```

2. String Data:

```
a3 = [ "one" "two" "three" "four" "five" "one" "three" "two" ];
b3 = [ "six" "seven" "one" ];
< c2,ind > = setunion(a3,b3);
print "C2=",c2;
print "Ind=",ind;
```

```
C2=
   |       1       2       3       4       5       6       7
-------------------------------------------------------------
 1 |    five    four     one   seven     six   three     two
```

```
Ind=
   |   1   2
-----------
 1 |   5  -1
 2 |   4  -1
 3 |   1   3
 4 |  -1   2
 5 |  -1   1
 6 |   3  -1
 7 |   2  -1
```

```
< c3,ind > = setunion(b3,a3);
print "C3=",c3;
print "Ind=",ind;
```

```
C3=
   |       1       2       3       4       5       6       7
-------------------------------------------------------------
 1 |    five    four     one   seven     six   three     two
```

```
Ind=
   |   1   2
------------
 1 |  -1   5
 2 |  -1   4
 3 |   3   1
 4 |   2  -1
 5 |   1  -1
 6 |  -1   3
 7 |  -1   2
```

3.9 Function `setxor`

< c,indx > = setxor(a,b)

Purpose: There are two different input versions:

1. Arguments `a` and `b` are scalars or vectors:
2. Arguments `a` and `b` are matrices with the same column number:

After reducing `a` and `b` to sets of unique entries (removing duplicates) this function returns in `c` the set XOR (exclusive or) of `a` with `b`.

Input: **a** can be scalar, vector, or matrix

b can be scalar, vector, or matrix

Output: **c** Returns the K unique entries which are either in `a` or in `b` but not in both (without replications). The entries of `c` are sorted with numeric values at first and string values last. If $K = 0$ a missing value is returned.

indx Returns the $K \times 2$ matrix indicating which of the entries of `c` is from `a` (first column) and which is from `b` (second column). For this function each row must contain one -1 value, indicating the entry can only come from one of the two sets.

Restrictions: 1. The input arguments can be numeric, including missing values, or string or of mixed data type.

2. The arguments `a`, and `b` can be scalar, vector, or matrix. If `a` or `b` is matrix, the other must be a matrix with the same number of columns.

Relationships: setdiff(), setisect(), setmembr(), setunion(), unique()

Examples: 1. Numerical Data:

```
a1 = [ 2#3 3#4 2 3 1 4 5 ];
a2 = [ 2#3. 3#4. . . . 2. 3. 1 4. 6. . . . ];
< c1,ind > = setxor(a1,a2);
print "C1=",c1;
print "Ind=",ind;
```

```
C1=
   |         1         2         3
---------------------------------
 1 |    .         5.0000    6.0000
```

```
Ind=
   |    1    2
--------------
 1 |   -1    6
 2 |   10   -1
 3 |   -1   13
```

2. String Data:

```
   a3 = [ "one" "two" "three" "four" "five" "one" "three" "two" ];
   b3 = [ "six" "seven" "one" ];
   < c2,ind > = setxor(a3,b3);
   print "C2=",c2;
   print "Ind=",ind;
```

```
C2=
   |       1       2       3       4       5       6
-------------------------------------------------------
 1 |    five    four   seven     six   three     two

Ind=
   |   1   2
------------
 1 |   5  -1
 2 |   4  -1
 3 |  -1   2
 4 |  -1   1
 5 |   3  -1
 6 |   2  -1
```

```
   < c3,ind > = setxor(b3,a3);
   print "C3=",c3;
   print "Ind=",ind;
```

```
C3=
   |       1       2       3       4       5       6
-------------------------------------------------------
 1 |    five    four   seven     six   three     two

Ind=
   |   1   2
------------
```

```
1 |  -1   5
2 |  -1   4
3 |   2  -1
4 |   1  -1
5 |  -1   3
6 |  -1   2
```

3.10 Function `size`

nd = size(a<,ind>)

Purpose: The function `size` returns either:

1. A vector of the size of the dimensions of a data object: For vectors and matrices it returns a 2-vector with the number of rows and columns. For tensors `size` returns a vector with more than 2 integers.
2. An integer of the size of a specific dimension which is specified by the second input argument.

This function is similar to the `size` function in Matlab.

Input: 1. The first argument of function `size` specifies a data object like scalar, vector, matrix, or tensor.

2. The second input argument is optional. If it is used it should be an integer within the scope of the dimensionality of the first argument.

Output: The function `size` has only one return which is either an integer vector or an integer scalar or a missing value.

Restrictions: 1. There are no obvious restrictions for the input object `a`.

2. Note, the second input argument depends on the specified index base, which is by default equal to one, but maybe changed to zero by the runtime option.

Relationships: nrow(), ncol(), dim()

Examples: 1. Vector and Matrix:

```
brv = [ 1 2 3 ];
dbr = size(brv);
print "Dim of row vector=",dbr;

bcv = [ 1, 2, 3 ];
dbc = size(bcv);
print "Dim of col vector=",dbc;

bmat = [ 100. 200. 300.,
         300. 200. 100. ];
db2 = size(bmat);
print "Dim of matrix=",db2;
```

```
Dim of row vector=
   |   1   2
-----------
 1 |   1   3

Dim of col vector=
   |   1   2
-----------
 1 |   3   1

Dim of matrix=
   |   1   2
-----------
 1 |   2   3
```

2. Tensor: Use of `size` without second input argument:

```
real B1[3,4,5];
print "B1[3,4,5]=", B1;
n1 = size(B1);
print "Dimension B1=", n1;
```

```
Dimension B1=
   |   1   2   3
---------------
 1 |   3   4   5
```

3. Tensor: Use of `size` with second input argument:

```
real B2[3,4,5] = 99.99;
print "B2[3,4,5]=", B2;
n3 = size(B2,3);
print "Number columns of B2=", n3;
```

```
Number columns of B2= 5
```

3.11 Function `sortrow`

< rank,bmat > = sortrow(amat<,key<,optn>>)

Purpose: The rows of a $N \times n$ matrix **A** are sorted w.r.t. the combined value of a sorting key composed from a specified (sub)set of the column of **A**. Applied to categorical variables (int columns) this function with `optn[3]=1` is useful also for obtaining multidimensional frequency tables.

Input: amat The first input argument is a n *timesn* **A**.

key The second (optional) argument is a vector of k ints specifiying the key columns that should be used for sorting.

optn The (optional) third argument is an option vector:

1. `optn[1]=1`: sort descending (default is ascending)
2. `optn[2]=1`: use the absolute values for the sorting key
3. `optn[3]=1`: result `bmat` is a matrix containing all distinct sorted key columns with a frequency column.

Output: rank is an N integer vector with the ranking of the original rows

bmat default: is the matrix **B** of the sorted data. If `optn[3]=1`: `bmat` is a matrix containing all distinct sorted key columns with a frequency column attached at the end.

Restrictions:
1. No missing values or string data are permitted in the sorting key columns of `amat`.
2. If the key columns contain complex values, its absolute value is used for sorting and ranking.

Relationships: branks(), ranktie(), subscript operators: <, <:, < |, <!, >, >:, > |, >!

Examples: 1. Small example: ASC_3Opt: nobs=429, nvar=5:

```
  options NOECHO;
  asc_3opt = [
#include "..\\tdata\\asc_3opt.dat"
           ];
  options ECHO;
  asc_3opt = shape(asc_3opt,.,5);
  cnam = [ "i1":"i5" ];
  asc_3opt = cname(asc_3opt,cnam);
```

Default Result:

```
  < irnk,dsort > = sortrow(asc_3opt);
  print "Irnk=", irnk;
  /* print "Dsort=", dsort; */
```

```
Irnk=
    |     1
----------
  1 |   281
  2 |   181
  3 |   224
  4 |   278
  5 |    79
  .........
425 |   292
426 |   354
427 |   134
428 |    28
429 |   429
```

This should be the same:

```
  key = [ 1:5 ];
  < irnk,dsort > = sortrow(asc_3opt,key);
  print "Irnk=", irnk;
  print "Dsort=", dsort;
```

Sort descending:

```
  < irnk,dsort > = sortrow(asc_3opt,.,1);
  print "Irnk=", irnk;
  print "Dsort=", dsort;
```

Alternative result: There are 101 distinct patterns in the key columns. The last column contains the frequency of the pattern.

```
  optn = [ ., ., 1];
  < irnk,keys > = sortrow(asc_3opt,.,optn);
  print "Irnk=", irnk;
  print "Dsort=", dsort;
```

```
Keys=
    |    1    2    3    4    5    6
-----------------------------------
  1 |    0    0    0    0    0    5
```

```
  2 |    0    0    0    1    0    1
  3 |    0    0    0    2    1    1
  4 |    0    0    1    0    0    4
  5 |    0    0    1    1    0    5
  6 |    0    0    1    2    1    1
  7 |    0    0    1    2    2    1
  8 |    0    0    2    1    0    2
  9 |    0    0    2    1    1    1
 10 |    0    0    2    2    0    1
 ..................................
 90 |    2    1    2    2    1   16
 91 |    2    1    2    2    2   11
 92 |    2    2    0    1    0    1
 93 |    2    2    1    0    2    1
 94 |    2    2    1    1    0    1
 95 |    2    2    1    1    1    1
 96 |    2    2    1    2    0    3
 97 |    2    2    2    1    0    3
 98 |    2    2    2    1    1    4
 99 |    2    2    2    1    2    5
100 |    2    2    2    2    0   11
101 |    2    2    2    2    1   22
```

2. Large example: SF36: nobs=2126, nvar=5:

```
  options NOECHO;
  sf36 = [
#include "..\\tdata\\sf36_mh.dat"
          ];
  options ECHO;
  sf36 = shape(sf36,.,5);
  cnam = [ "sfnerv" "sfdump" "sfcalm" "sflow" "sfhappy" ];
  sf36 = cname(sf36,cnam);
```

Alternative Result:

```
  optn = [ ., ., 1];
  < irnk,keys > = sortrow(sf36,.,optn);
  print "Irnk=", irnk;
  print "Dsort=", dsort;
```

```
Irnk=
     |      1
-------------
   1 |    797
```

```
   2 |    134
   3 |   1099
   4 |      4
   5 |   1599
   6 |   1267
   7 |    803
   8 |   1203
   9 |   1469
  10 |     76
  ...........
2120 |   2114
2121 |   1977
2122 |    669
2123 |   1369
2124 |   1737
2125 |   1787
2126 |    998
```

There are 419 distinct patterns in the key columns. The last column contains the frequency of the pattern.

Keys=

```
    |     1     2     3     4     5     6
-------------------------------------------
  1 |     0     0     0     0     0    88
  2 |     0     0     0     0     1    41
  3 |     0     0     0     0     2     1
  4 |     0     0     0     0     4     3
  5 |     0     0     0     1     0     3
  6 |     0     0     0     1     1     5
  7 |     0     0     0     2     0     3
  8 |     0     0     0     2     1     1
  9 |     0     0     0     4     0     1
 10 |     0     0     1     0     0    56
 ...........................................
410 |     4     4     1     2     1     1
411 |     4     4     1     3     0     1
412 |     4     4     1     3     2     1
413 |     4     4     1     4     0     1
414 |     4     4     1     4     1     1
415 |     4     4     2     1     3     1
416 |     4     4     2     3     3     1
417 |     4     4     3     3     4     1
418 |     4     4     4     2     3     1
419 |     4     4     4     4     1     1
```

Default Result:

```
< irnk,dsort > = sortrow(sf36);
print "Irnk=", irnk;
```

3.12 Function `unique`

< c,i1,i2 > = unique(a¡,fuzzy¿)

Purpose: There are two different input versions:

1. Argument `a` is scalar or vector:
2. Argument `a` is matrix.

The input argument `a` is reduced to a set of unique entries which is returned in `c`. The entries of `c` are sorted with the numerical values at the begin (a missing value would be in the first location) and string data at the end.

Input: **a** can be scalar, vector, or matrix

b can be scalar, vector, or matrix

Output: **c** Returns the K the unique entries of `a`.

i1 index vector defining `c = a[i1]`

i2 index vector defining `a = c[i2]`

Restrictions: 1. The input arguments can be numeric, including missing values, or string or of mixed data type.

Relationships: setdiff(), setisect(), setmembr(), setxor(), setunion()

Examples: 1. Numerical Data:

```
a1 = [ 2#3 3#4 2 3 1 4 5 ];
< b1,ind,jnd > = unique(a1);
print "B1=",b1;
print "ind=",ind;
print "jnd=",jnd;
```

```
B1=
   |   1   2   3   4   5
-----------------------
 1 |   1   2   3   4   5
```

```
ind=
   |    1
---------
 1 |    8
 2 |    6
 3 |    2
 4 |    4
 5 |   10
```

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

```
a2 = [ 2#3. 3#4 . . . 2 3 1 4. 5 . . . ];
< b2,ind,jnd > = unique(a2);
print "B2=",b2;
print "ind=",ind;
print "jnd=",jnd;
```

```
B2=
   |         1         2         3         4         5         6
-------------------------------------------------------------------
 1 |    .        1.00000    2.0000    3.0000    4.0000    5.0000
```

```
ind=
   |    1
---------
 1 |    6
 2 |   11
 3 |    9
 4 |   10
 5 |    3
 6 |   13
```

```
jnd=
   |   1
--------
 1 |   4
 2 |   4
 3 |   5
 4 |   5
 5 |   5
 6 |   1
 7 |   1
 8 |   1
 9 |   3
10 |   4
11 |   2
12 |   5
13 |   6
14 |   1
15 |   1
16 |   1
```

2. String Data:

```
a3 = [ "one" "two" "three" "four" "five" "one" "three" "two" ];
< b3,ind,jnd > = unique(a3);
print "B3=",b3;
print "ind=",ind;
print "jnd=",jnd;
```

```
B3=
   |       1       2       3       4       5
-------------------------------------------------
 1 |    five    four     one   three     two


ind=
   |   1
--------
 1 |   5
 2 |   4
 3 |   1
 4 |   3
 5 |   2


jnd=
   |   1
--------
 1 |   3
 2 |   5
 3 |   4
 4 |   2
 5 |   1
 6 |   3
 7 |   4
 8 |   5
```

```
   a4 = [ "six" "seven" "one" ];
   < b4,ind,jnd > = unique(a4);
   print "B4=",b4;
   print "ind=",ind;
   print "jnd=",jnd;
```

```
B4=
   |       1       2       3
---------------------------
 1 |     one   seven     six
```

```
ind=                                    jnd=
   |   1                                   |   1
--------                                --------
 1 |   3                                  1 |   3
 2 |   2                                  2 |   2
 3 |   1                                  3 |   1
```

3.13 Function `vec2tri`

b = vec2tri(a<,nc<,opt>>)

Purpose: This function moves the entries of an input vector `a` into either

1. a lower triangular matrix `b`,
2. an upper triangular matrix `b`,
3. a symmetric matrix `b`.

Input: **a** The first input argument must be a n vector, $n > 1$.

nc The second input argument specifies the number of columns of the matrix `b`. For lower triangular and symmetric matrix this argument is optional. When the result is specified upper triangular this argument must be specified.

"sopt" the third input argument is optional. If used it must have one of the following string values:

"dlow" vector `a` specifies the values of the lower triangle of `b`.

"slow" vector `a` specifies the values of the strict lower triangle of `b`. The diagonal of `b` is set to zero.

"dupp" vector `a` specifies the values of the upper triangle of `b`.

"supp" vector `a` specifies the values of the strict lower triangle of `b`. The diagonal of `b` is set to zero.

"dsym" vector `a` specifies the values of the lower triangle of the symmetric matrix `b`.

"ssym" vector `a` specifies the values of the strict lower triangle of the symmetric matrix `b`. The diagonal of `b` is set to zero.

Default is `"dlow"`.

Output: The result `b` is either a lower or upper triangular or symmetric matrix containing the values of the vector `a`.

Restrictions: 1. Input argument `a` cannot be a scalar.

Relationships: tri2vec(), shape(), tri2sym

Examples: 1.