CMAT Newsletter: July 2005

Wolfgang M. Hartmann

July 2005

Contents

1 General Remarks

The `==` and `!=` operators now also permit missing values. For example `c = (a == .) ? 1 : 0;` should now work properly.
The `svd()` function was extended in two important ways:

1. By specifying an integer argument p the truncated `svd` of an $m \times n$ matrix $\mathbf{A}$ can now be computed,

$$\mathbf{A} = \mathbf{USV}^T \quad \text{where} \quad \mathbf{S} = Diag$$

   where only the largest $p < MIN(m, n)$ singular values and corresponding singular vectors are computed. When $p \ll MIN(m, n)$ this may save much computer time. The same was already possible when using the `svdtrip()` or the `arpack()` functions.

2. Modifications of the classic least squares SVD by some forms of alternating linear regressions can now be computed. All methods available in the `reg` function can be used for the iterative alternating estimation process, which include L_1, L_∞, L_p, for power $p \geq 1$ (but small), the robust LMS and the (fast) LTS methods as well as the orthogonal (total least squares) and some Huber estimation methods. When using the L_1 method the *robust SVD* by Liu, Hawkins, Ghosh, & Young (2003) results. This approach makes it possible to compute the SVD in the presence of missing values in the data matrix $\mathbf{A}$.

Both, the `pca()` and the `noharm()` functions now have some more return arguments and they are able to perform orthogonal and oblique Procrustes (target rotation) with two different algorithms. Also, the `frotate()` now is tested for Procrustes (target) rotation.
Two fast interior point methods where added to solve the linear least squares problems with

1. nonnegativity constraints
2. or general bound (box) constraints.

Therefore, the function `lls` was extended to have five arguments, `lls(`$a, b, sopt <$ `,`$optn <, bc >>$`)` for $m \times n$ matrix $\mathbf{A}$. Each of the two algorithms can take advantage of large sparse matrices $\mathbf{A}$ and could be more efficient than solving this problem using the more general `qp()` function. Here `bc` is either a n vector specifying lower bounds or a $n \times 2$ matrix specifying lower bounds in the first column and upper bounds in its second column.
The centroid method for the factorization of a matrix (Chu and Funderlik, 2002; Thurstone, 1931) and the non-negative matrix factorization (Lee and Seung, 1999, 2001; Hoyer, 2004; Shanaz et.al., 2004) were added.
Bootstrap methods were also added to both, the `factor()` and `noharm()` functions for ASEs and confidence intervals of

- goodness of fit indices,
- of rotated or unrotated factor loadings, unique variances,
- and in case of oblique factor rotation also factor correlations and factor structure.

See below for new options added to `factor()` and `noharm()`.
Bootstrap methods were added to the `pca()` function for ASEs and confidence intervals of

- variance accounted for (VAF),
- the eigenvalues,
- and for rotated or unrotated component loadings.

See below for new options added to `pca()`.
A number of new functions were added. The last section, *Illustration*, shows some examples for bootstrap with the `noharm` function for factor analysis with binary data. In addition a rather long and not so easy *CMAT* macro was written for some applications of the bivariate Dale (BDM) model, where there is an ordinally scaled bivariate response.

1.1 New Functions

The following new functions are implemented:

centroid implements two algorithms for computing the centroid factorization of a given rectangular real matrix **A** which is an approximation to the singular value decomposition or the principal component analysis.

generead input of micro array data or comma separated data sets.

histogrm compute k histogram for each column of a given data matrix **A**.

impute impute missing values into an object **A**. This function is not completely finished yet.

nnmf implements several algorithms for computing the non-negative matrix factorization of a given rectangular real matrix **A** with nonnegative entries.

permute compute stepwise combinations (with replications) and permutations.

quantile compute k quantiles for each column of a given data matrix **A**.

simdid compute $n \times n$ matrix of various similarity measures of n histograms based on n columns of data matrix **A**.

svdupd computes a rank r update of the SVD (U, S, V) of the $m \times n$ matrix **A**. This is not a computationally efficient version yet.

1.2 Fixed Bugs

1. The most important bug: the `a = spmat(`$nr, nc, rind, cind, val <, sopt >$`)` function did not work when the `val` vector (argument number 5) was not of type real.

2. Some of the return arguments for `factor` were not filled correctly.

3. The `sprintf` function did return for short strings some trash fillup at the end of the result. This was fixed.

4. An important bug was found in the *SIMPLS* algorithm which is part of the `pls` algorithm.

5. A terrible bug was fixed with the ranking functions like the >! index operation.

6. A number of bugs in the `glim` function were fixed.

7. The `qp` did return the starting vector `x0` when there were no constraints specified. Now it shiuld return the unconstrained optimal solution.

2 Modifications of Features

2.1 Extensions to `svd()` Function

There are two extensions:

1. The second argument of the `svm` function may be specified as an integer p, where $p < \min(m, n)$, if the *truncated* SVD with only a small number p of singular values and vectors is needed. This option can save much computer time and memory. See also the `svdtrip` function for some similar approaches more designed for sparse matrices $\mathbf{A}$.

2. An additional third argument was needed to offer some variations of the common least squares SVD. For specified $p < \min(m, n)$ some alternating linear regression methods are now available for *robust* SVD and for the EM estimation in the presence of missing values in $m \times n$ matrix $\mathbf{A}$.

When using the L_1 method the *robust SVD* by Liu, Hawkins, Ghosh, & Young (2003) results. As a nice side effect, this approach makes it possible to compute the SVD in the presence of missing values in the data matrix $\mathbf{A}$. The missing values are estimated in an EM like fashion.
The function now has the following syntax:

<sval, v, u> = svd(a<,"eco"—p<optn>>)

Some of the following options are identical to those of the `reg()` function:

Option	Second Column	Meaning
"cent"		center all variables w.r.t. mean
"cstrt"		compute starting values using the centroid method
"fast"		only in combination with LTS estimation: use "Fast LTS" by Rousseeuw and Huber
"lpow"	real	power value p used in L_p regression
"maxi"	int	maximum number of iterations
"meth"	string	linear regression method
	"l_2"	least squares (L_2) estimation
	"l_1"	least absolute value (LAV, L_1) estimation
	"l_i"	Chebyshev regression, L_∞ estimation
	"l_p"	general L_p norm regression, where $p \geq 1$
	"odi"	linear orthogonal distance regression, total least squares (TLS), errors-in-variables regression (EIV)
	"hub"	robust (Huber) regression where *large* residuals enter the objective function with smaller weights than small residuals r_i.
	"lms"	resistant least median squares regression
	"lts"	resistant least trimmed squares regression
"print"	int	integer value specifying the amount of general printed output (=0: no output)
"quant"	int	quantile $q > 0$ used in the objective function, for $q = 0$, default $q = (m + n + 1)/2$ is used.
"scal"		scale all variables w.r.t. standard deviation
"seed"	int	seed value for random generator used for generating initial values
"sstrt"		compute starting values with common SVD
"tol"	real	function termination criterion for iteration
"vers"	string	version of algorithm:
	"ort_w"	only for `l_2`: orthogonalize left matrix **W**
	"ort_h"	only for `l_2`: orthogonalize right matrix **H**
	"dtls"	only for `odi`: use DTLS algorithm
	"ptls"	only for `odi`: use PTLS algorithm
	"mani"	only for `l_1`: use Madsen & Nielsen algorithm
	"baro"	only for `l_1`: use Barrodale & Roberts algorithm
	"pine"	only for `l_i`: use Pinar & Elhedlhi algorithm
	"baph"	only for `l_i`: use Barrodale & Philips algorithm
	"arku"	only for `l_i`: use Armstrong & Kung algorithm
	"and"	only for Huber: Andrews
	"biw"	only for Huber: Biweight
	"cau"	only for Huber: Cauchy
	"fai"	only for Huber: Fair
	"hub"	only for Huber: Huber
	"log"	only for Huber: Logistic
	"tal"	only for Huber: Talwar
	"wel"	only for Huber: Welsch

Some cautious remarks about the estimation methods:

1. Alternating linear regression seems to work (converge) well for L_2, L_1, L_∞, and Huber estimation when the input data have no missing values.

2. Convergence problems may have to do with different scaled data. That means, we found examples where the algorithm did not converge for unscaled data, but was converging for standardized data. This is especially so when the data have too many missing values.

3. The algorithm works well (converges) for L_p regression when $1 \leq p \leq 2$. For power $p \gg 2$ it may not converge. There maybe is a fix point theorem which could show that behavior. Obviously, a modification may not bring much improvement.

4. For total least squares (orthogonal regression) it does not converge currently, but it may be possible to modify the algorithm using a different normalization.

5. The algorithm seem to work well for LTS. It seems to take more iterations to converge for LMS, but there seems to be some problems to converge for the current form of the *Fast-LTS* algorithm. Maybe there is still a bug left.

The following example uses the *European Health and Fertility Data* by Croux, Filzmoser, Pison, Rousseeuw (2004). The columns correspond to the following variables:

1. average population growth from 1986-2000
2. women in the age able to give birth in % (1985)
3. proportion of women per 100 men (1985)
4. life expectancy of women (1986)
5. life expectancy of men (1986)
6. infant mortality rate (1986)
7. inhabitants per doctor (1981)
8. daily calorie consumption per head (1985)
9. proportion of babies with underweight at birth in % (1984)

The observations (cases) corespond to the following countries:

1. Austria
2. Albania

3. Bulgaria
4. Switzerland
5. Czechoslowakia
6. DDR (East Germany)
7. Hungary
8. Norway
9. Poland
10. Romania
11. Sweden
12. Finland
13. SovietUnion
14. Turkey
15. Yugoslavia
16. European Community (averaged across member countries until 1986)

```
euro = [  -0.1    48    110    77    70    10    440    3440     6,
           1.8    50     97    75    68    41   2100    2716     7,
           0.2    47    101    75    69    15    400    3593     6,
           0.0    44    103    80    74     7    390    3406     5,
           0.3    46    105    75    66    14    350    3473     6,
           0.0    47    110    75    68     9    490    3769     6,
          -0.1    46    106    75    67    19    390    3544    10,
           0.2    48    101    80    74     9    460    3171     4,
           0.6    48    104    76    68    18    550    3224     8,
           0.5    47    102    73    68    26    700    3413     6,
           0.0    47    101    80    74     6    410    3007     4,
           0.2    47    107    79    72     6    460    2961     4,
           0.7    48    112    73    64    30    270    3332     6,
           1.9    49     97    67    62    79   1530    3218     8,
           0.5    51    103    74    68    27    700    3499     7,
          0.22   48.4  103.9  78.3  72.6  10.2  509.1  3420.5   5.4 ];

 cnam = [ "popgrow" "womibag" "propw2m" "lifexw" "lifexm" "infmort"
          "inhpdoc" "calcons" "uwgtbab" ];
 rnam = [ "Austria" "Albania" "Bulgaria" "Switzer" "Czechos"
          "DDR"     "Hungary" "Norway"   "Poland"  "Romania"
          "Sweden"  "Finland" "SovietU"  "Turkey"  "Yugosla"
          "EC" ];
```

```
   euro = cname(euro,cnam);
   euro = rname(euro,rnam);

  /* common SVD: */
  < q0,v0,u0 > = svd(euro,2);
  print "Q0=", q0;
  print "V0=", v0;
  print "SSQ=",ssq(euro - u0 * q0 * v0‘);

Q0=
 D |          1          2
---------------------------
 1 |   13578.20          0
 2 |          0    1962.44

V0=
          |         1         2
--------------------------------
  popgrow |   0.00012   0.00108
  womibag |   0.01399   0.00384
  propw2m |   0.03058  -0.00127
   lifexw |   0.02225   0.00043
   lifexm |   0.02027   0.00081
  infmort |   0.00601   0.02708
  inhpdoc |   0.18427   0.98243
  calcons |   0.98182  -0.18459
  uwgtbab |   0.00182   0.00112

SSQ= 4666.79

  /* Alternating ALS estimation w/o Orthogonalization */
  optn = [ "print"         2 ,
           "meth"      "l_2" ,
           "seed"        123 ];
  < q1,v1,u1 > = svd(euro,2,optn);
  print "Q1=", q1;
  print "V1=", v1;
  print "V1'*V1=", v1‘ * v1;
  print "U1'*U1'", u1‘ * u1;

                Alternating Linear ALS Regression

         Iter      L2Crit    DiffCrit      L1Crit     LooCrit

            1  5285.91418   188218117  500.461321  36.7098221
            2  4666.78575  619.128430  479.447952  33.5970248
            3  4666.78551  2.395e-004  479.435632  33.6034757
```

```
            4  4666.78551  1.692e-010  479.435619  33.6034806

          L2 Precision : 4666.79   L1 Precision : 479.436
                     Computer Time : 0 seconds

Q1=
 D |         1         2
-----------------------
 1 |   8308.25         0
 2 |         0   8035.77

V1=
          |         1         2
--------------------------------
  popgrow |  -0.00012   0.00038
  womibag |   0.01276   0.01450
  propw2m |   0.03008   0.02941
   lifexw |   0.02158   0.02172
   lifexm |   0.01957   0.01989
  infmort |  -0.00025   0.01223
  inhpdoc |  -0.04180   0.41098
  calcons |   0.99817   0.91050
  uwgtbab |   0.00152   0.00203

V1'*V1=
 S |         1         2
-----------------------
 1 |   1.00000
 2 |   0.89358   1.00000

U1'*U1=
 S |         1         2
-----------------------
 1 |   1.00000
 2 |   0.45776   1.00000

  /* Alternating ALS estimation with H Orthogonalization */
  optn = [ "print"         2 ,
           "meth"      "l_2" ,
           "vers"    "ort_h" ,
           "seed"        123 ];
  < q1,v1,u1 > = svd(euro,2,optn);
  print "Q1=", q1;
  print "V1=", v1;
  print "V1'*V1=", v1' * v1;
  print "U1'*U1=", u1' * u1;
```

```
                Alternating Linear ALS Regression
               Column Orthogonalization of H Matrix

         Iter      L2Crit    DiffCrit      L1Crit     LooCrit

            1  5285.91418   188218117  500.461321  36.7098221
            2  4666.78575  619.128430  479.447952  33.5970248
            3  4666.78551  2.395e-004  479.435632  33.6034757
            4  4666.78551  1.701e-010  479.435619  33.6034806

          L2 Precision : 4666.79   L1 Precision : 479.436
                    Computer Time : 0 seconds

SSQ= 4666.79

Q1=
 D |          1          2
---------------------------
 1 |   13236.53          0
 2 |          0    3607.36

V1=
          |         1         2
--------------------------------
  popgrow |  -0.00012   0.00108
  womibag |   0.01276   0.00689
  propw2m |   0.03008   0.00565
   lifexw |   0.02158   0.00543
   lifexm |   0.01957   0.00536
  infmort |  -0.00025   0.02774
  inhpdoc |  -0.04180   0.99869
  calcons |   0.99817   0.04134
  uwgtbab |   0.00152   0.00150

V1'*V1=
 S |         1         2
------------------------
 1 |   1.00000
 2 |    5e-019   1.00000

U1'*U1
 S |         1         2
------------------------
 1 |   1.00000
 2 |   0.82981   1.00000

  /* Alternating LAV estimation: Madsen-Nielsen */
```

```
  optn = [ "print"         2 ,
           "meth"      "l_1" ,
           "vers"     "mani" ,
           "seed"        123 ];
  < q2,v2,u2 > = svd(euro,2,optn);
  print "Q2=", q2;
  print "V2=", v2;

                Alternating Linear LAV Regression
                   Algorithm : Madsen & Nielsen

         Iter      L1Crit    DiffCrit      L2Crit     LooCrit

            1  886.605748  67622.1143  25030.2558  66.8357772
            2  447.699453  438.906294  5499.97052  46.2314961
            3  447.699453  2.046e-012  5499.97052  46.2314961

          L2 Precision : 5499.97   L1 Precision : 447.699
                    Computer Time : 1 seconds

Q2=
 D |          1          2
---------------------------
 1 |   10647.78          0
 2 |          0   10417.21

V2=
           |         1         2
-------------------------------
  popgrow |  -0.00001   0.00022
  womibag |   0.01348   0.01454
  propw2m |   0.02974   0.03039
   lifexw |   0.02172   0.02272
   lifexm |   0.01902   0.02012
  infmort |   0.00342   0.00728
  inhpdoc |   0.06577   0.30638
  calcons |   0.99687   0.95080
  uwgtbab |   0.00160   0.00186

  /* Alternating L_infty estimation: Pinar-Elhedlhi */
  optn = [ "print"         2 ,
           "meth"      "l_i" ,
           "vers"     "pine" ,
           "seed"        123 ];
  < q2,v2,u2 > = svd(euro,2,optn);
  print "Q2=", q2;
```

```
                Alternating Linear L_inf Regression
                    Algorithm : Pinar & Elhedlhi

         Iter     LooCrit    DiffCrit      L1Crit      L2Crit

            1  62.7742514  3706.22575  1305.67695  30825.9093
            2  23.7663920  39.0078595  1190.65525  18268.2178
            3  23.5986815  0.16771044  1187.36375  18086.8195
            4  23.5986815  9.095e-013  1187.36477  18086.8203

          L2 Precision : 18086.8   L1 Precision : 1187.36
                     Computer Time : 0 seconds

Q2=
 D |          1          2
---------------------------
 1 |   10512.16          0
 2 |          0    5126.52

   /* Alternating L_p */
   optn = [ "print"         2 ,
            "meth"      "l_p" ,
            "lpow"        1.5 ,
            "seed"        123 ];
    < q2,v2,u2 > = svd(euro,2,optn);

              Alternating Lp Estimation with Power 1.5

         Iter      L2Crit    DiffCrit      L1Crit     LooCrit

            1  164.236145  22524.2175  597.154203  48.8302208
            2  122.937471  41.2986736  462.043129  37.4142422
            3  122.937449  2.206e-005  462.039017  37.4186886

          L2 Precision : 4759.18   L1 Precision : 462.039
                     Computer Time : 0 seconds

   /* Alternating Huber: Biweight */
   optn = [ "print"         2 ,
            "meth"      "hub" ,
            "vers"      "biw" ,
            "seed"        123 ];
     < q2,v2,u2 > = svd(euro,2,optn);

              Alternating Lp Estimation with Power 2.5

         Iter      L2Crit    DiffCrit      L1Crit     LooCrit
```

```
          1  55.8170723  10209.7609  530.997900  40.5964945
          2  50.0355272  5.78154516  509.029570  31.4010393
          3  50.0355304 -3.294e-006  509.030577  31.4032680

       L2 Precision : 4734.42   L1 Precision : 509.031
                  Computer Time : 0 seconds
```

```
/* Alternating LTS: Least Trimmed Squared */
optn = [ "print"         2 ,
         "meth"      "lts" ,
         "seed"        123 ];
< q2,v2,u2 > = svd(euro,2,optn);
```

```
              Alternating Linear LTS Regression

      Iter      L2Crit    DiffCrit      L1Crit     LooCrit

         1  20454974.9           .  8337.14030  3465.78409
         2  3802620.42  16652354.5  6604.38583  1243.22758
         3  550574.960  3252045.46  1915.96503  491.225879
         4  137259.096  413315.865  785.232279  248.270547
         5  412720.390  275461.294  2349.56016  307.171998
         6  2394332.71  1981612.32  4634.65377  902.878871
         7  3196.06888  2391136.64  105.357923  44.5414708
         8  4.836e-005  3196.06883  0.00965090  0.00577060
         9  3.390e-017  4.836e-005  1.291e-008  4.520e-009

    L2 Precision : 4.88098e+006   L1 Precision : 7392.67
                  Computer Time : 0 seconds
```

```
/* Alternating LMS: Least Median Squared */
optn = [ "print"         2 ,
         "meth"      "lms" ,
         "seed"        123 ];
< q2,v2,u2 > = svd(euro,2,optn);
```

```
              Alternating Linear LMS Regression

      Iter      L2Crit    DiffCrit      L1Crit     LooCrit

         1  20699795.6           .  8678.61370  3008.13866
         2  10140479.4  10559316.2  9265.49152  2121.28336
         3  1240866.92  8899612.43  3564.24932  818.139205
         4  215705.012  1025161.91  1177.07742  377.009711
         5  36013.6908  179691.322  612.622355  95.9867867
         6  3316.47901  32697.2118  170.545538  36.1233801
```

```
 7  3241505.91  3238189.43  3126.67875  1557.62502
 8  52482.5408  3189023.37  760.868294  155.631350
 9  35560.4959  16922.0449  402.333599  149.515720
10  829139.330  793578.834  1531.68366  802.558431
11  10081.3457  819057.985  240.037323  66.1510902
12  247253.443  237172.097  962.800259  399.420791
13  193688.357  53565.0859  1336.04469  232.595739
14  544340.261  350651.904  1427.44408  667.285983
15  15102.7719  529237.489  403.251427  74.2577344
16  1027939.25  1012836.48  2609.50068  748.136339
17  910358.593  117580.661  2670.37551  536.586224
18  575193.425  335165.168  2255.66098  471.905441
19  1370714.20  795520.776  2905.61945  871.515904
20  34027.6472  1336686.55  645.424280  101.078811
21  868070.108  834042.461  1345.59800  823.418452
22  36211.9204  831858.187  738.794432  86.1960249
23  1105067.48  1068855.56  1888.89052  902.203176
24  20771.5102  1084295.97  380.576055  84.8043965
25  14775.6151  5995.89504  211.797906  84.8059701
26  184.042064  14591.5731  21.2459899  11.8359748
27  8.739e-022  184.042064  1.282e-010  1.182e-011
28  3.822e-022  4.917e-022  6.844e-011  1.347e-011

L2 Precision : 4.90208e+006   L1 Precision : 6373.75
              Computer Time : 0 seconds
```

2.2 Extensions to `svm()` Function

2.3 Extensions to `lls()` Function

Two very fast interior point methods where added to solve the linear least squares problems with

1. nonnegativity constraints or
2. general bound (box) constraints.

Therefore, the function `lls` was extended to have five arguments, `lls(`$a, b, sopt <$ `,`$optn <, bc >>$`)` for $m \times n$ matrix $\mathbf{A}$. Each of the two algorithms can take advantage of large sparse matrices $\mathbf{A}$ and could be more efficient than solving this problem using the more general `qp()` function. Here `bc` is either a n vector specifying lower bounds or a $n \times 2$ matrix specifying lower bounds in the first column and upper bounds in its second column.

Some Examples

1. Using the `bc` argument: Data set with $m = 29, n = 2$:

```
  wdat= [ 0.9248    0.8504,   0.9991     0.297,     0.608    0.8464,
          0.7149    0.6585,   0.249     0.3328,   0.09659    0.6194,
           0.267    0.3359,   0.7851     0.486,    0.3631    0.4496,
          0.8838    0.1325,   0.4412   0.06382,    0.7928    0.4361,
          0.3789    0.2249,   0.1726    0.9518,    0.9129    0.8851,
          0.6418     0.533,   0.6561    0.6456,    0.9483    0.4861,
          0.9552    0.3621,   0.3976    0.8404,    0.8326    0.3555,
          0.4939    0.6732,   0.5024    0.8439,    0.2477    0.6779,
          0.5566     0.166,   0.8805    0.3252,   0.02409    0.6227,
          0.2674    0.9978,   0.8742    0.6924 ];

  b = [  0   1   0   0   0   0   0   0   0   0
         0   0   0   0   0   0   0   0   0   0
         0   0   0   0   1   1   0   0   1 ]';

  /* Unconstrained LS Problem */
  x = lls(wdat,b,"gel");
  print "LLS X=",x, " L2 error=", sqrt(ssq(wdat*x - b));

LLS X=
   |         1
--------------
 1 |   0.42632
 2 |  -0.21616
 L2 error= 1.7055

  /* Corresponding QP */
  S = wdat' * wdat; v = -wdat' * b;
  x0 = [ .5 .5 ];
  optn = [ "print"      5 ];
  < xr, rp > = qp(S,v,x0,optn);
  print "LLS X=",xr, " L2 error=", sqrt(ssq(wdat*xr' - b));

LLS X=
   |         1         2
------------------------
 1 |   0.42632  -0.21616
 L2 error= 1.7055

  /* Nonnegative LS Problem */
  bc = [ 0., 0.];
  optn = [ "print"     2 ,
           "maxit"   100 ];
  x = lls(wdat,b,"gel",optn,bc);
  print "Nonnegative X=",x;
```

```
                Lower and Upper Bound Constraints
                *********************************

                      Dense Column Vector bc

                   C |           1          2
                         0.0000000  0.0000000

                 Nonnegative Linear Least Squares

   Iter   NH1   NH2    NF    NG        C\_2        C\_1       C\_oo

      0     0     2     0     2  2.00000000  4.00000000  1.00000000
      1     1     0     2     0  1.70551069  6.60308982  0.79859455
      2     0     0     1     1  1.76160472  6.77589586  0.84922354

                 L2 Computer Time : 0 seconds

Nonnegative X=
   |         1
--------------
 1 |   0.27089
 2 |   0.00000

  /* Corresponding QP */
  S = wdat' * wdat; v = -wdat' * b;
  x0 = [ .5 .5 ];
  optn = [ "print"      3 ,
           "bounds"  "bc" ];
  < xr, rp > = qp(S,v,x0,optn);
  print "LLS X=",xr, " L2 error=", sqrt(ssq(wdat*xr' - b));

                 TRON (Lin and More) QP Method
                      Using Dense Hessian


    Iteration Start:
    N. Variables             2
    N. Bound. Constr.        2       N. Mask  Constr.          0
    Criterion       2.638926213       Max Grad Entry  8.153282308
    N. Active Constraints    0

   Iter  nfun    optcrit     actred    prered     gnorm     delta
      1     2  0.00e+000     2.6389    2.6389    3.6264    0.7071
      2     4    -0.3427     0.3427    0.3427    1.6071    0.7071
```

```
      3     6    -0.4484     0.1057    0.1057 4.44e-016    0.7071
      4     8    -0.4484  5.55e-017 8.07e-033 4.44e-016    0.7071
            Relative function convergence 2.22045e-016.

   Successful Termination After     4 Iterations
   FCONV convergence criterion satisfied.
   Criterion      -0.448374398        Max Grad Entry  4.4409e-016
   N. Active Constraints     1
   N. Function Calls        11        N. Gradient Calls        10
   N. Line Searches         21        Preproces. Time           0
   Time for Method           1        Effective Time            1

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

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

           Parameter     Estimate   Gradient   Active BC

            1 X1       0.27088835 -4.44e-016
            2 X2       0.00000000  0.8997263  Lower BC

            Value of Objective Function =    -0.448374

LLS X=
   |         1         2
------------------------
 1 |   0.27089   0.00000
 L2 error= 1.7616

  /* Box Constrained LS Problem */
  bc = [ 0. 100. , 0. 100 ];
  optn = [ "print"      2 ,
           "tol"    1.e-8 ,
           "maxit"    100 ];
  x = lls(wdat,b,"gel",optn,bc);
  print "Box Constrained Nonnegative X=",x;

                Lower and Upper Bound Constraints
                *********************************

                         Dense Matrix bc
```

```
                      |    Lower BC   Upper BC
                      -------------------------
                      1 |   0.0000000  100.00000
                      2 |   0.0000000  100.00000

            Linear Least Squares with Box Constraints

     Iter      Stop1      Stop2        Gap       C\_2       C\_1

        1  0.0000000  5.39e-013  0.1601290  1.8350526  7.4088423
        2  0.0000000  7.98e-014  0.0201080  1.7616269  6.8018086
        3  0.0000000  3.28e-016  1.46e-006  1.7616048  6.7759006
        4  0.0000000  5.63e-016  7.28e-013  1.7616047  6.7758959

                  L2 Computer Time : 0 seconds

Box Constrained Nonnegative X=
   |         1
--------------
 1 |   0.27089
 2 |    4e-014

  /* Corresponding QP */
  S = wdat' * wdat; v = -wdat' * b;
  x0 = [ .5 .5 ];
  optn = [ "print"      3 ,
           "bounds"  "bc" ];
  < xr, rp > = qp(S,v,x0,optn);
  print "LLS X=",xr, " L2 error=", sqrt(ssq(wdat*xr' - b));

                  TRON (Lin and More) QP Method
                       Using Dense Hessian


    Iteration Start:
    N. Variables             2
    N. Bound. Constr.        4        N. Mask  Constr.          0
    Criterion       2.638926213       Max Grad Entry  8.153282308
    N. Active Constraints    0

   Iter  nfun    optcrit     actred    prered     gnorm     delta
      1     2  0.00e+000     2.6389    2.6389    3.6264    0.7071
      2     4    -0.3427     0.3427    0.3427    1.6071    0.7071
      3     6    -0.4484     0.1057    0.1057 4.44e-016    0.7071
      4     8    -0.4484  5.55e-017 8.07e-033 4.44e-016    0.7071
           Relative function convergence 2.22045e-016.
```

```
Successful Termination After     4 Iterations
FCONV convergence criterion satisfied.
Criterion      -0.448374398        Max Grad Entry  4.4409e-016
N. Active Constraints     1
N. Function Calls        11        N. Gradient Calls        10
N. Line Searches         21        Preproces. Time           0
Time for Method           0        Effective Time            0

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

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

          Parameter     Estimate   Gradient   Active BC

           1 X1       0.27088835 -4.44e-016
           2 X2       0.00000000  0.8997263  Lower BC

           Value of Objective Function =    -0.448374

LLS X=
   |         1         2
------------------------
 1 |   0.27089   0.00000
 L2 error= 1.7616
```

2. Australian data set: $m = 690, n = 15$:

```
form = fo = " %g";
for (j = 2; j <= 15; j++) form = strcat(form,fo);
fid = fopen("..\\tdata\\australian.dat","r");
data = fscanf(fid,form,.,15);
nr = nrow(data);
print "Observations of Australian.dat:",nr;
x = data[,2:15]; y = data[,1];

/* Australian: nobs=690, nvar=15 */
form = fo = " %g";
for (j = 2; j <= 15; j++) form = strcat(form,fo);
fid = fopen("..\\tdata\\australian.dat","r");
data = fscanf(fid,form,.,15);
nr = nrow(data);
print "Observations of Australian.dat:",nr;
x = data[,2:15]; y = data[,1];
```

```
  /* Unconstrained LS Problem */
  sol = lls(x,y,"gel");
  print "Unconstrained LLS X=",sol;

Unconstrained LLS X=

   |         1
-------------
 1 |   0.00294
 2 |  -0.00332
 3 |   0.04218
 4 |   0.00174
 5 |   0.01825
 6 |   0.01337
 7 |   0.01929
 8 |  -0.08990
 9 |   0.00155
10 |   0.04257
11 |   0.19255
12 |   0.00022
13 |   0.00000
14 |  -0.04011

  /* Corresponding QP */
  S = x' * x; v = -x' * y;
  x0 = cons(1,14,.5);
  optn = [ "print"      3 ];
  < xr, rp > = qp(S,v,x0,optn);
  print "LLS X=",xr, " L2 error=", sqrt(ssq(x*xr' - y));

LLS X=
   |         1         2         3         4         5
-----------------------------------------------------
 1 |   0.00294  -0.00332   0.04218   0.00174   0.01825

   |         6         7         8         9        10
-----------------------------------------------------
 1 |   0.01337   0.01929  -0.08990   0.00155   0.04257

   |        11        12        13        14
-------------------------------------------
 1 |   0.19255   0.00022   0.00000  -0.04011
 L2 error= 12.355

  /* Nonnegative LS Problem */
```

```
bc = cons(14,1,0.);
optn = [ "print"     2 ,
         "maxit"   100 ];
sol = lls(x,y,"gel",optn,bc);
print "Nonnegative X=",sol;
```

Lower and Upper Bound Constraints
****

Dense Column Vector bc

C	1	2	3	4	5
	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

C	6	7	8	9	10
	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

C	11	12	13	14
	0.0000000	0.0000000	0.0000000	0.0000000

Nonnegative Linear Least Squares

Iter	NH1	NH2	NF	NG	C_2	C_1	C_oo
0	0	14	0	14	21.6333077	468.000000	1.00000000
1	4	0	14	0	12.3552053	297.659928	1.11929906
2	2	0	10	4	12.4043834	299.695073	1.03932847
3	1	0	8	6	12.4212083	300.214349	1.03931242
4	0	0	7	7	12.4215932	300.257734	1.03728805

L2 Computer Time : 0 seconds

Nonnegative X=

	1
1	0.00308
2	0.00000
3	0.02871
4	0.00000
5	0.01865
6	0.00904
7	0.00000
8	0.00000
9	0.00000
10	0.04554
11	0.17782
12	0.00025

```
13 |   0.00000
14 |   0.00000

  /* Corresponding QP */
  S = x' * x; v = -x' * y;
  x0 = cons(1,14,.5);
  optn = [ "print"      3 ,
           "bounds"  "bc" ];
  < xr, rp > = qp(S,v,x0,optn);
  print "LLS X=",xr, " L2 error=", sqrt(ssq(x*xr' - y));

                  TRON (Lin and More) QP Method
                       Using Dense Hessian

    Iteration Start:
    N. Variables             14
    N. Bound. Constr.        14        N. Mask  Constr.          0
    Criterion        2489009440        Max Grad Entry   9817617986
    N. Active Constraints     0

   Iter  nfun    optcrit     actred    prered     gnorm     delta
      1     2  0.00e+000  2.49e+009 2.49e+009 4.91e+005    1.8708
      2     4   -35.4073    35.4073   35.4073 4.23e+005    1.8708
      3     6  -105.9178    70.5105   70.5105 2.36e+005    1.8708
      4     8  -125.7437    19.8259   19.8259 1.97e+005    1.8708
      5    10  -146.1749    20.4312   20.4312 62198.4480    1.8708
      6    12  -154.3604     8.1855    8.1855 67659.0117    1.8708
      7    14  -155.8332     1.4729    1.4729 5949.0002    1.8708
      8    16  -156.8520     1.0188    1.0188 1.82e-012    1.8708
      9    18  -156.8520  2.84e-014 7.22e-029 1.84e-012    1.8708
           Relative function convergence 2.22045e-016.

    Successful Termination After     9 Iterations
    FCONV convergence criterion satisfied.
    Criterion      -156.8520114        Max Grad Entry  1.8190e-012
    N. Active Constraints     7
    N. Function Calls        21        N. Gradient Calls        20
    N. Line Searches         76        Preproces. Time           0
    Time for Method           0        Effective Time            0

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

                         Parameter Estimates
```

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

           Parameter      Estimate   Gradient   Active BC

             1 X1       0.00307548 -1.59e-012
             2 X2       0.00000000  60.120770  Lower BC
             3 X3       0.02871396  7.11e-014
             4 X4       0.00000000  8.5047467  Lower BC
             5 X5       0.01865139  8.53e-014
             6 X6       0.00903707  2.42e-013
             7 X7       0.00000000  5.8668597  Lower BC
             8 X8       0.00000000  14.082777  Lower BC
             9 X9       0.00000000  66.173670  Lower BC
            10 X10      0.04554393  3.55e-015
            11 X11      0.17781913 -2.84e-014
            12 X12      2.545e-004  1.46e-011
            13 X13      0.00000000  45915.917  Lower BC
            14 X14      0.00000000  8.6712550  Lower BC

            Value of Objective Function =     -156.852

LLS X=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |   0.00308   0.00000   0.02871   0.00000   0.01865

   |         6         7         8         9        10
---------------------------------------------------------
 1 |   0.00904   0.00000   0.00000   0.00000   0.04554

   |        11        12        13        14
---------------------------------------------
 1 |   0.17782   0.00025   0.00000   0.00000
 L2 error= 12.422

  /* Box Constrained LS Problem */
  bc = cons(14,1,0.) -> cons(14,1,10.);
  optn = [ "print"      2 ,
           "tol"    1.e-6 ,
           "maxit"    100 ];
  sol = lls(x,y,"gel",optn,bc);
  print "Box Constrained Nonnegative X=",sol;

             Linear Least Squares with Box Constraints

     Iter      Stop1      Stop2        Gap       C\_2       C\_1
```

```
 1  0.0000000  1.54e-005  4117.4928  65.318060  601.86225
 2  3.55e-015  1.53e-005  4111.9388  65.139391  597.53927
 3  4.35e-015  1.45e-005  4179.9128  63.378189  562.26861
 4  3.97e-015  8.86e-006  279.89462  17.345942  346.59390
 5  3.97e-015  4.40e-006  203.26030  15.315274  320.38787
 6  1.78e-015  1.14e-007  31.184117  13.077902  306.95571
 7  3.55e-015  3.06e-008  33.403165  13.018561  301.06169
 8  3.97e-015  1.82e-008  16.726691  12.683849  300.08113
 9  3.08e-015  5.66e-009  2.5767571  12.458319  300.47795
10  3.08e-015  5.48e-011  0.1443392  12.426411  300.32433
11  2.51e-015  1.46e-011  0.0027826  12.421761  300.26460
12  2.51e-015  7.28e-011  3.64e-005  12.421596  300.25787
13  3.97e-015  9.46e-011  1.15e-010  12.421593  300.25773

                L2 Computer Time : 0 seconds

Box Constrained Nonnegative X=

   |         1
---------------
 1 |   0.00308
 2 |    7e-014
 3 |   0.02871
 4 |    2e-012
 5 |   0.01865
 6 |   0.00904
 7 |    1e-012
 8 |    3e-013
 9 |    9e-014
10 |   0.04554
11 |   0.17782
12 |   0.00025
13 |    2e-016
14 |    7e-013

  /* Corresponding QP */
  S = x‘ * x; v = -x‘ * y;
  x0 = cons(1,14,.5);
  optn = [ "print"      3 ,
           "bounds"  "bc" ];
  < xr, rp > = qp(S,v,x0,optn);
  print "LLS X=",xr, " L2 error=", sqrt(ssq(x*xr‘ - y));

                TRON (Lin and More) QP Method
                     Using Dense Hessian
```

```
 Iteration Start:
 N. Variables             14
 N. Bound. Constr.        28        N. Mask  Constr.          0
 Criterion        2489009440        Max Grad Entry   9817617986
 N. Active Constraints     0

Iter  nfun    optcrit     actred    prered     gnorm     delta
   1     2  0.00e+000  2.49e+009 2.49e+009 4.91e+005    1.8708
   2     4   -35.4073    35.4073   35.4073 4.23e+005    1.8708
   3     6  -105.9178    70.5105   70.5105 2.36e+005    1.8708
   4     8  -125.7437    19.8259   19.8259 1.97e+005    1.8708
   5    10  -146.1749    20.4312   20.4312 62198.4480    1.8708
   6    12  -154.3604     8.1855    8.1855 67659.0117    1.8708
   7    14  -155.8332     1.4729    1.4729 5949.0002    1.8708
   8    16  -156.8520     1.0188    1.0188 1.82e-012    1.8708
   9    18  -156.8520  2.84e-014 7.22e-029 1.84e-012    1.8708
        Relative function convergence 2.22045e-016.

 Successful Termination After     9 Iterations
 FCONV convergence criterion satisfied.
 Criterion      -156.8520114        Max Grad Entry  1.8190e-012
 N. Active Constraints     7
 N. Function Calls        21        N. Gradient Calls        20
 N. Line Searches         76        Preproces. Time           0
 Time for Method           0        Effective Time            0

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

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

       Parameter      Estimate   Gradient   Active BC

         1 X1       0.00307548 -1.59e-012
         2 X2       0.00000000  60.120770  Lower BC
         3 X3       0.02871396  7.11e-014
         4 X4       0.00000000  8.5047467  Lower BC
         5 X5       0.01865139  8.53e-014
         6 X6       0.00903707  2.42e-013
         7 X7       0.00000000  5.8668597  Lower BC
         8 X8       0.00000000  14.082777  Lower BC
         9 X9       0.00000000  66.173670  Lower BC
        10 X10      0.04554393  3.55e-015
```

```
            11 X11      0.17781913 -2.84e-014
            12 X12      2.545e-004  1.46e-011
            13 X13      0.00000000  45915.917  Lower BC
            14 X14      0.00000000  8.6712550  Lower BC

            Value of Objective Function =     -156.852

LLS X=
   |         1         2         3         4         5
---------------------------------------------------------
 1 |   0.00308   0.00000   0.02871   0.00000   0.01865

   |         6         7         8         9        10
---------------------------------------------------------
 1 |   0.00904   0.00000   0.00000   0.00000   0.04554

   |        11        12        13        14
-----------------------------------------------
 1 |   0.17782   0.00025   0.00000   0.00000
 L2 error= 12.422
```

2.4 Extensions to `spmat()` Function

The function `a = spmat(`$nr, nc, rind, cind, val <, sopt >$`)` was extended to other data type in the `val` argument. It will now also work for mixed, i.e. string and numeric data type. In the past it was working only for real data type.

2.5 Extensions to `factor()`and `noharm()` Functions

For the `noharm()` function now the entire suite of orthogonal and oblique factor rotations is implemented which was already available for the `factor` and `frotate` functions.

<gof,parm,resi,covm,boci> = noharm(data,optn<,guess<,init<,targ<,wtrg>>>>)

Bootstrap methods were added to the `factor()` and `noharm()` functions for confidence intervals of goodness of fit indices, for rotated or unrotated factor loadings, unique variances (only for `factor`), and in case of oblique factor rotation also factor correlations and factor structure. See below for new options added to `factor()`.

Option	Second Column	Meaning
"bala"	int	=1: balanced (default) or =0:uniform resampling
	string	"y": yes, "n": no
		(same as in jboot and sem
"bcorr"		bias correction for skewness
"bgof"		compute bootstrap confidence intervals for goodness of fit values
"boci"		compute bootstrap confidence intervals for unique vars and rotated or unrotated loadings
"btask"	string	bootstrap method
	"jack"	Jackknife is performed (fast)
	"norm"	normal theory bootstrap (fast)
	"hybr"	hybrid boostrap (fast)
	"perc"	percetile bootstrap (fast)
	"bc"	bias corrected bootstrap (fast)
	"bca"	bias accelerated corrected bootstrap
	"stud"	studentized bootstrap (only for "boci")
	"all"	perform all bootstrap methods
		(same as in jboot and sem
		default is perc for "gbof" and "bc" otherwise
"bseed"	int	seed value for random generator
		(same as in jboot and sem
"cl"	string	type of confidence intervals
	"none"	do not compute
	"wald"	Wald linkelihood intervals (fast)
	"boot"	Bootstrap confidence intervals (slow)
"rand"	string	specifying the random generator ("ranu": RAND Corporation, "kiss": keep-it-simple-stupid, "lecu": by L'Ecuyer ([?])
		(same as in jboot and sem
"samp"	int	number of bootstrap samples (default=200)
		(same as in jboot and sem

2.6 Extensions to `pca()` Function

Among the many additions to the `pca()` function belong the new features of target rotation and bootstrap ASEs and CIs. Due to a number of changes we list here the entire document for the revised `pca` function.

<gof,eval,evec,flod,frot,prot,boci> = pca(data,optn<,targ<,wtrg>>)

Purpose: The `pca` function implements a number of algorithms for principal component analysis. Assuming a $N \times n$ data matrix $\mathbf{X}$ with $n \times n$ covariance or correlation matrix $\mathbf{C}$,

$$\mathbf{C} = \mathbf{L}\mathbf{L}^T$$

and $\mathbf{L}$ is the *ntimesm* component loading matrix, usually in the form $\mathbf{L} = \mathbf{V}\mathbf{D}^{\mathbf{1/2}}$ where $\mathbf{V}$ is the *ntimesm* eigenvector matrix with m orthogonal columns and $\mathbf{D}$ is the $m \times m$ diagonal matrix containing the m largest eigenvalues of $\mathbf{C}$. Using the *ver*sion option we can select an algorithm among the eight available implementations. The first four compute eigenvalue decompositions of symmetric cross product (covariance or correlation) matrices (either $\mathtt{X}^T\mathbf{X}$ or $\mathtt{X}\mathtt{X}^T$ whichever is the smaller one) and the other four methods use the singular value decomposition of the raw matrix $\mathbf{X}$:

"cev1" (11) all values, all vectors dense eigen value decomposition (EVD): the algorithm is based on the eigenvalue decomposition of a dense symmetric X^TX or XX^T matrix and can be applied to raw input data and also to $n \times n$ input covariance or correlation matrices;

"cev2" (12) all values, few vectors dense EVD: the algorithm is based on the eigenvalue decomposition of a dense symmetric X^TX or XX^T matrix and can be applied to raw input data and also to $n \times n$ input covariance or correlation matrices;

"cev3" (13) selected triplets Lapack EVD: the algorithm is based on the eigenvalue decomposition of a dense symmetric X^TX or XX^T matrix and can be applied to raw input data and also to $n \times n$ input covariance or correlation matrices;

"cev4" (14) selected triplets Arpack EVD: the algorithm is appropriate for large and sparse data matrix $\mathbf{X}$ and is based on the eigenvalue decomposition of a dense symmetric X^TX or XX^T matrix and can be applied to raw input data and also to $n \times n$ input covariance or correlation matrices;

"rsv1" (16) standard dense singular value decomposition (SVD): the algorithm is suitable for medium sized dense data matrices $\mathbf{X}$, and cannot be applied to $n \times n$ input covariance or correlation matrices;

"rsv2" (17) selected triplets Arpack SVD: the algorithm is suitable for large sized sparse data matrices $\mathbf{X}$ and a relatively small number of

factors, and cannot be applied to $n \times n$ input covariance or correlation matrices;

"rsv3" (18) selected triplets Block Lanczos SVD: the algorithm is suitable for large sized sparse data matrices $\mathbf{X}$, and cannot be applied to $n \times n$ input covariance or correlation matrices;

"rsv4" (19) selected triplets subspace iteration SVD: the algorithm is suitable for large sized sparse data matrices $\mathbf{X}$, and cannot be applied to $n \times n$ input covariance or correlation matrices.

The eigenvalue algorithms need memory in $O(n^2) + O(n * m)$ or $O(N^2) + O(N * m) + O(n * m)$ whereas the singular value algorithms need memory $O(N * m) + O(n * m)$ for dense $\mathbf{X}$ and $O(nzer) + O(n * m)$ for sparse $\mathbf{X}$. There are a number of criteria for defining the number of components (factors) which can be selected by the following options:

"nfac" This option requires an integer specifying the number components (factors).

"mineig" This option requires a real value for a lower bound of the smallest eigenvalue for the component

"perc" This option requires an int or real value for the proportion of common variance to be accounted for by the selected components. Values in $[0, 1]$ specify a relative proportion, values in $[1, 100]$ a percentage. Default is 1 or 100 %.

"stud" The scree test with the studendized residual is used for specifying the number of factors and requires a real input value in $(0, 1)$ for α (default: $\alpha = .1$).

"cook" The scree test with Cooks distance is used for specifying the number of factors and requires a real input value in $(0, 1)$ for the threshold (default threshold is .25).

For each of these except the `"nfac"` option, the prior computation of a set of eigenvalues is necessary. Note, that the `"perc"`, `"stud"`, and `"cook"` options require the computation of all eigenvalues. This is no problem for dense and medium sized data sets, but maybe a computational problem for very large data sets. When only the `"mineig"` option is specified only the required subset of eigenvalues is computed and sparsity of the matrix can be exploited.

The asymptotic standard errors and Wald confidence intervals of rotated factor loadings depend on:

1. the kind of rotation method appplied and especially the fact whether the rotation is orthogonal or oblique;
2. if the (unstandardized) covariance or the (standardized) correlation matrix is analysed. (Note, you may read in a correlation matrix, but if you specify the analysis option `"anal"` as `"cov"`, the results correspond to those of a covariance analysis.)

3. if Kaiser normalization of the loadings is applied for the specific rotation method.

Note, the asymptotic standard errors and confidence intervals are available only with algorithms `cev1,...,cev4` for $N > n$, i.e. when a $n \times n$ correlation or covariance matrix is computed and stored. They are also currently not available for PROMAX rotated components.

Input: data The first argument `data` must be the name of a data object specifiying a

1. `nobs` by `nvar` matrix of raw data: all algorithms can be applied
2. symmetric `nvar` by `nvar` matrix of covariances or correlations.
3. `nvar+1` by `nvar` matrix that contains a symmetric covariance or correlation matrix in its first `nvar` rows and a vector of mean values in its last row.

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.

targ This argument specifies the target matrix for Browne's Procrustes rotation method. This is used for both, `"targt1"` and `"targt2"` rotation. It must be of dimension $nvar \times nfact$.

wtrg This argument specifies the weight matrix for the target matrix of Browne's Procrustes rotation method. This is used only for `"targt2"` rotation. It must be of dimension $nvar \times nfact$.

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

Option	Second Column	Meaning
"alpha"	real	the significance level for confidence intervals; default is $\alpha = 0.05$
"ana"	string	data type for analysis
	"cor"	correlations are analyzed
	"cov"	covariances are analyzed
	"ucor"	uncorrected correlations are analyzed
	"ucov"	uncorrected covariances are analyzed
"bala"	int	=1: balanced (default) or =0:uniform resampling
	string	"y": yes, "n": no
		(same as in jboot and sem
"bcorr"		bias correction for skewness
"bgof"		compute bootstrap confidence intervals for goodness of fit values
"boci"		compute bootstrap confidence intervals for unique vars and rotated or unrotated loadings
"btask"	string	bootstrap method
	"jack"	Jackknife is performed (fast)
	"norm"	normal theory bootstrap (fast)
	"hybr"	hybrid boostrap (fast)
	"perc"	percetile bootstrap (fast)
	"bc"	bias corrected bootstrap (fast)
	"bca"	bias accelerated corrected bootstrap
	"stud"	studentized bootstrap (only for "boci")
	"all"	perform all bootstrap methods
		(same as in jboot and sem
		default is perc for "gbof" and "bc" otherwise
"bseed"	int	seed value for random generator
		(same as in jboot and sem
"cl"	string	type of confidence intervals
	"none"	do not compute
	"wald"	Wald linkelihood intervals (fast)
	"boot"	Bootstrap confidence intervals (slow)
"cook"	real	use the scree test with Cooks distance
		for specifying the number of factors
		default threshold is .25
"data"	string	type of input data
	"raw"	$N \times n$ raw data
	"cor"	symmetric correlation matrix
	"cov"	symmetric covariance matrix
"freq"	int	column number of FREQ variable when raw data input

Option	Second Column	Meaning
"frot"	string	rotation method for exploratory FACTOR model
	"none"	no rotation
	"crafer"	orthogonal Crawford-Ferguson family (specify γ)
	"varmax"	orthogonal Varimax rotation (crafer with $\gamma = \frac{1}{n}$)
	"quamax"	orthogonal Quartimax rotation (crafer with $\gamma = 0$)
	"equmax"	orthogonal Equamax rotation (crafer with $\gamma = \frac{q}{2n}$
	"parmax"	orthogonal Parsimax rotation (crafer with $\gamma = \frac{q-1}{n+q-2}$)
	"parmax"	orthogonal Factor Parsimony rotation (crafer with $\gamma = 1$)
	"bentlr"	orthogonal Bentler rotation criterion
	"minent"	orthogonal Minimum Entropy rotation
	"oblmin"	orthogonal Direct Oblimin family (specify γ)
	"simmax"	orthogonal Simplimax (specify number nonzero loadings with γ)
	"tandm1"	orthogonal Tandem 1 rotation
	"tandm2"	orthogonal Tandem 2 rotation
	"infmax"	orthogonal Infomax rotation
	"mccamm"	orthogonal McCammon rotation
	"crafer"	oblique Crawford-Ferguson family (specify γ)
	"oblmin"	oblique Direct Oblimin family (specify γ)
	"quamin"	oblique Direct Quartimin (oblmin with $\gamma = 0$)
	"biqmin"	oblique Bi-Quartimin (oblmin with $\gamma = .5$)
	"covmin"	oblique Covarimin (oblmin with $\gamma = 1$)
	"bentlr"	oblique Bentler rotation criterion
	"simmax"	oblique Simplimax (specify number nonzero loadings with γ)
	"oblmax"	oblique Oblimax
	"geomin"	oblique Geomin
	"infmax"	oblique Infomax rotation
	"promax"	oblique Promax
	"targt1"	Target Rotation (Harman, 1976)
	"targt2"	Weighted Target (Browne,2001)
"fnorm"		Kaiser's factor pattern normalization use only for exploratory FACTOR model
"mineig"	real	smallest eigenvalue for which factor is retained used for specifying the number of factors
"nfac"	int	number of components (factors)

Option	Second Column	Meaning
"nobs"	real	number observations when COV or CORR data input (is mostly an integer)
"nopr"		perform no printed output
"prin"	int	amount of printed output (=0 is noprint)
"pall"		large amount of printed output
"phis"		some technical output on the performance of algorithms
"perc"	real	proportion of common variance to be accounted for used for specifying the number of factors default is 1 or 100 %
"psho"		short amount of printed output
"psum"		summary amount of printed output
"rand"	string	specifying the random generator ("ranu": RAND Corporation, "kiss": keep-it-simple-stupid, "lecu": by L'Ecuyer ([**?**]) (same as in `jboot` and `sem`
"ridge"	real	nonnegative ridge value
"rinit"	int	initialization for rotation matrix $\mathbf{T}$
"rotp"	real	rotation parameter (γ for Crawford-Ferguson)
"rvers"	int	version of algorithm for rotation
"samp"	int	number of bootstrap samples (default=200) (same as in `jboot` and `sem`
"seed"	real	seed value for random generator
"sing"	real	singularity threshold (default: 1.e-8)
"stud"	real	use the scree test with studendized residual for specifying the number of factors default: $\alpha = .1$
"vers"		version of algorithm
	"cev1" (11)	all values, all vectors dense EVD: column oriented
	"cev2" (12)	all values, few vectors dense EVD: column oriented
	"cev3" (13)	selected triplets Lapack EVD: column oriented
	"cev4" (14)	selected triplets Arpack EVD: column oriented
	"rsv1" (16)	standard dense SVD: row oriented
	"rsv2" (17)	selected triplets Arpack SVD: row oriented
	"rsv3" (18)	selected triplets Block Lanczos SVD: row oriented
	"rsv4" (19)	selected triplets Rutishauser-Ritz SVD: row oriented
"wgt"	int	column number of WEIGHT variable when raw data input

The number of components (factors) is specified either using the `nfac`, the `mineig` or the `perc` option.

The following table shows whether orthogonal or oblique rotation is the default and if the default can be changed by specifying `par[6]`:

spec	Rotation Method	orthogonal	oblique
"quamax"	Quartimax (Carroll, 1953)	default	yes
"varmax"	Varimax (Kaiser, 1958)	default	yes
"equmax"	Equamax	default	yes
"parmax"	Parsimax	default	yes
"facpar"	Factor Parsimony	default	yes
"crafer"	Crawford-Ferguson (1970)	default	yes
"minent"	Minimum Entropy (Jennrich)	default	no
"tandm1"	Tandem 1 (Comrey,1967)	default	no
"tandm2"	Tandem 2 (Comrey, 1967)	default	no
"mccamm"	McCammon (1966)	default	no
"quamin"	Quartimin (Carroll, 1953)	yes	default
"simmax"	Simplimax (Kiers, 1994)	yes	default
"oblmin"	Oblimin (Carroll, 1960)	yes	default
"biqmin"	Bi-Quartimin	yes	default
"covmin"	Covarimin	yes	default
"bentlr"	Bentler Criterion (1977)	yes	default
"infmax"	Infomax (McKeon,1968)	yes	default
"oblmax"	Oblimax (Saunders,1961)	no	default
"geomin"	Geomin (Yates, 1984)	no	default
"promax"	Promax (Hendrickson,White)	no	default
"targt1"	Target Rotation (Harman,1976)	yes	default
"targt2"	Weighted Target (Browne,2001)	yes	default

Output: **gof** column vector with goodness-of-fit measures. For specified bootstrap, the vector becomes a matrix.

eval vector of m largest eigenvalues

evec the $n \times m$ matrix $\mathbf{V}$ of eigenvectors containing orthogonal eigenvectors in m columns

flod the $n \times mk$ matrix of unrotated components where $mk = m$ if no Wald ASEs are computed and $mk = 4{*}m$ if Wald ASEs are computed

frot the $n \times mk$ matrix of rotated components where $mk = m$ if no Wald ASEs are computed and $mk = 4 * m$ if Wald ASEs are computed

prot the $m \times m$ matrix of component correlations

boci contains bootstrap confidence intervals of (rotated or unrotated) component loadings, and in the case of oblique rotation also component correlations.

Restrictions: 1. The input data cannot contain any missing or string data.

2.

Relationships: noharm(), sem(), frotate(), factor()

Examples:

Compare Different Methods :

1. Australian data set: N=690, n=15:
   This data set is tall and skinny:

```
form = fo = " %g";
for (j = 2; j <= 15; j++) form = strcat(form,fo);
fid = fopen("..\\tdata\\australian.dat","r");
data = fscanf(fid,form,.,15);
nr = nrow(data);
print "Observations of Australian.dat:",nr;
x = data[,2:15]; y = data[,1];


optn = [ "data"      "raw" ,
         "anal"      "cov" ,
         "vers"     "cev1" ,
         "nfac"          2 ,
         "frot"   "varmax" ,
         "prin"          3 ];
< gof,eval > = pca(data,optn);


                  ***************************
                  Principal Component Analysis
                  ***************************

  Input Data. . . . . . . . . . . . . . . . . . . . . . Raw Data
  Analysis of . . . . . . . . . . . . . . . . . Covariance Matrix
  Number of Items . . . . . . . . . . . . . . . . . .        15
  Number of Factors (NFactor Criterion) . . . . . .         2
  Number of Subjects. . . . . . . . . . . . . . . . .      690
  C-Version . . . . . . . . . . . . . .Standard Full EVD Code
  Orthogonal Rotation Method. . . . . .Varimax (Kaiser, 1958)
  Unnormed Rotation: Parameter. . . . . . . . . . . 1.0000000


  Name             Mean     Std Dev    Skewness    Kurtosis
  ------------------------------------------------------------
  V_1       0.678260870  0.46748239 -0.76485905 -1.41911243
  V_2       31.56820290  11.8532728  1.15593502  1.19205859
  V_3       4.758724635  4.97816324  1.48881311  2.27402175
  V_4       1.766666667  0.43006283 -1.15345501 -0.33826957
  V_5       7.372463768  3.68326479 -0.06919047 -0.84904258
```

V_6	4.692753623	1.99231607	0.46841183	-0.17813233
V_7	2.223405797	3.34651336	2.89133042	11.2001917
V_8	0.523188406	0.49982433	-0.09305595	-1.99713783
V_9	0.427536232	0.49508002	0.29358655	-1.91937879
V_10	2.400000000	4.86294003	5.15251986	50.8294313
V_11	0.457971014	0.49859186	0.16908083	-1.97715098
V_12	1.928985507	0.29881306	-1.94472547	6.71888763
V_13	184.0144928	172.159274	2.74991175	19.9266976
V_14	1018.385507	5210.10260	13.1406550	214.669972
V_15	0.444927536	0.49731827	0.22212157	-1.95634100

Mardia's Multivariate Kurtosis	382.6470
Relative Multivariate Kurtosis	2.5006
Normalized Multivariate Kurtosis	222.5398
Mardia Based Kappa (Browne, 1982).	1.5006
Mean Scaled Univariate Kurtosis	6.5817
Adjusted Mean Scaled Univariate Kurtosis . . .	6.7595

Observation numbers with largest contribution to kurtosis

501	183	364	234	150
85970.27	42631.19	10507.05	10466.83	6095.615

Eigenvalues of Model Covariance Matrix

Total = 27175027.0492 Average = 1811668.4699

	1	2	3	4
Eigenvalue	27145297.4	29513.8586	143.549416	29.2552327
Difference	27115783.5	29370.3091	114.294183	11.3534716
Proportion	0.99890599	0.00108607	5.282e-006	1.077e-006
Cumulative	0.99890599	0.99999206	0.99999734	0.99999842

	5	6	7	8
Eigenvalue	17.9017611	13.2933763	7.61042879	2.96541195
Difference	4.60838476	5.68294754	4.64501685	2.65464859
Proportion	6.588e-007	4.892e-007	2.801e-007	1.091e-007
Cumulative	0.99999908	0.99999957	0.99999985	0.99999996

	9	10	11	12
Eigenvalue	0.31076336	0.23923626	0.21373686	0.17137565
Difference	0.07152710	0.02549940	0.04236121	0.03691017
Proportion	1.144e-008	8.804e-009	7.865e-009	6.306e-009

Cumulative	0.99999997	0.99999998	0.99999998	0.99999999

	13	14	15
Eigenvalue	0.13446548	0.08216729	0.06420234
Difference	0.05229819	0.01796495	
Proportion	4.948e-009	3.024e-009	2.363e-009
Cumulative	0.99999999	1.00000000	1.00000000

Eigenvectors

	V_1	V_2	V_3	V_4	V_5
FAC_1	-3.25e-007	-4.22e-005	-1.18e-004	-9.91e-006	-2.17e-005
FAC_2	-1.59e-004	0.0054498	0.0066998	2.43e-005	-0.0018514

Eigenvectors

	V_6	V_7	V_8	V_9	V_10
FAC_1	-2.48e-005	-3.30e-005	-8.63e-006	-7.38e-006	-5.94e-005
FAC_2	-7.71e-004	0.0015620	2.14e-004	1.69e-004	0.0035233

Eigenvectors

	V_11	V_12	V_13	V_14	V_15
FAC_1	-1.84e-006	-8.01e-006	-0.0021703	-0.9999976	-1.68e-005
FAC_2	-4.16e-004	1.54e-004	-0.9999507	0.0021690	3.24e-004

Unrotated Factor Loadings

	FAC_1	FAC_2
V_1	0.0016922	0.0273332
V_2	0.2196791	-0.9362548
V_3	0.6128357	-1.1509950
V_4	0.0516352	-0.0041802
V_5	0.1132290	0.3180616
V_6	0.1291930	0.1325307
V_7	0.1718077	-0.2683385
V_8	0.0449876	-0.0367902
V_9	0.0384418	-0.0290412
V_10	0.3096898	-0.6052958
V_11	0.0095788	0.0714325
V_12	0.0417346	-0.0264962
V_13	11.307462	171.78751
V_14	5210.1026	-0.3726186

```
V_15   0.0873536 -0.0556346


Rotated Factor Loadings: Varimax (Kaiser, 1958)

           FAC_1      FAC_2
V_1    0.0016901  0.0273334
V_2    0.2197509 -0.9362379
V_3    0.6129240 -1.1509480
V_4    0.0516355 -0.0041763
V_5    0.1132046  0.3180703
V_6    0.1291829  0.1325406
V_7    0.1718283 -0.2683253
V_8    0.0449904 -0.0367868
V_9    0.0384441 -0.0290382
V_10   0.3097363 -0.6052720
V_11   0.0095733  0.0714332
V_12   0.0417366 -0.0264930
V_13   11.294286  171.78838
V_14   5210.1026  0.0269857
V_15   0.0873579 -0.0556279


Orthogonal Transformation Matrix T

             FAC_1      FAC_2
FAC_1   1.0000000  7.67e-005
FAC_2  -7.67e-005  1.0000000
```

The other methods `cev2`, `cev3`, and `cev4` differ only by the number of available eigenvalues. Also, not much different is the output of the R-SVD methods.

2. NIR Spectra data set: N=21, n=268

   This data set is small and fat:

```
  options NOECHO;
#include "..\\tdata\\nir.dat"
  options ECHO;
  nrtrn = nrow(xtrn); nc = ncol(xtrn);
  nrtst = nrow(xtst);
  print "nrtrn,nctrn=",nrtrn,nc;


  optn = [ "data"      "raw" ,
           "anal"      "cov" ,
           "vers"     "cev1" ,
```

```
        "nfac"          2 ,
        "frot"   "varmax" ,
        "prin"          3 ];
< gof,eval > = pca(xtrn,optn);


                ****************************
                Principal Component Analysis
                ****************************

  Input Data. . . . . . . . . . . . . . . . . . . . . . Raw Data
  Analysis of . . . . . . . . . . . . . . . . Covariance Matrix
  Number of Items . . . . . . . . . . . . . . . . . . .      268
  Number of Factors (NFactor Criterion) . . . . . . . .        2
  Number of Subjects. . . . . . . . . . . . . . . . . .       21
  C-Version . . . . . . . . . . . . . . .Standard Full EVD Code
  Orthogonal Rotation Method. . . . . .Varimax (Kaiser, 1958)
  Unnormed Rotation: Parameter. . . . . . . . . . . . 1.0000000

  Name             Mean     Std Dev    Skewness    Kurtosis
  ------------------------------------------------------------
  V_1       3.079785714  0.06108026 -1.81706398  6.45840428
  V_2       3.051352381  0.10994384 -2.31330971  6.11300328
  V_3       2.987742857  0.17333722 -1.93558666  3.68046775
  V_4       2.884128571  0.23204478 -1.49044496  1.77660072
  V_5       2.742685714  0.27304037 -1.18269186  0.74915327
  V_6       2.588371429  0.29988459 -0.96103697  0.20859643
  V_7       2.446547619  0.32125377 -0.73261085 -0.12784500
  V_8       2.327895238  0.34133601 -0.44372802 -0.38651980
  V_9       2.239533333  0.36645360 -0.13895768 -0.59896976
  V_10      2.181271429  0.40388758  0.08763269 -0.77947816
  ............................................................
  V_265     0.325914762  0.01662340  0.34193210  0.19795115
  V_266     0.325785714  0.01669600  0.31636266  0.18260176
  V_267     0.325888571  0.01673901  0.30450867  0.16884682
  V_268     0.326099048  0.01677756  0.30085481  0.16075352
```

This is only the first part of the 21×21 cross product matrix:

```
            Cross Products of the Transposed Data

               1          2          3          4          5
  1  1.1097232
  2  0.7505248  0.5809512
  3  0.9091725  0.5863159  0.7762054
```

4	0.3701252	0.3828433	0.2458641	0.3658969	
5	0.5046231	0.3720574	0.4122105	0.2202073	0.2496557

Eigenvalues of Model Covariance Matrix
Total = 12.6025 Average = 0.6001

	1	2	3	4
Eigenvalue	6.56002901	5.88880063	0.09221895	0.02818838
Difference	0.67122838	5.79658168	0.06403057	0.00848365
Proportion	0.52053381	0.46727229	0.00731751	0.00223673
Cumulative	0.52053381	0.98780610	0.99512361	0.99736034

	5	6	7	8
Eigenvalue	0.01970473	0.01078044	0.00109392	6.298e-004
Difference	0.00892429	0.00968652	4.641e-004	3.571e-004
Proportion	0.00156356	8.554e-004	8.680e-005	4.998e-005
Cumulative	0.99892390	0.99977932	0.99986612	0.99991609

	9	10	11	12
Eigenvalue	2.727e-004	2.412e-004	1.351e-004	1.149e-004
Difference	3.148e-005	1.061e-004	2.018e-005	3.837e-005
Proportion	2.164e-005	1.914e-005	1.072e-005	9.116e-006
Cumulative	0.99993773	0.99995687	0.99996759	0.99997671

	13	14	15	16
Eigenvalue	7.652e-005	6.770e-005	5.081e-005	3.921e-005
Difference	8.823e-006	1.689e-005	1.160e-005	1.451e-005
Proportion	6.072e-006	5.372e-006	4.031e-006	3.111e-006
Cumulative	0.99998278	0.99998815	0.99999218	0.99999529

	17	18	19	20
Eigenvalue	2.470e-005	1.674e-005	1.041e-005	7.467e-006
Difference	7.955e-006	6.329e-006	2.946e-006	7.467e-006
Proportion	1.960e-006	1.328e-006	8.262e-007	5.925e-007
Cumulative	0.99999725	0.99999858	0.99999941	1.00000000

	21
Eigenvalue	-5.690e-017
Difference	
Proportion	-4.515e-018
Cumulative	1.00000000

Eigenvectors

	1	2	3	4	5
FAC_1	-0.1450285	-0.0029272	-0.1659867	0.1284992	-0.0327733
FAC_2	0.4004388	0.3123114	0.3172391	0.2075480	0.2023632

Eigenvectors

	6	7	8	9	10
FAC_1	-0.2207129	0.2737133	0.0974562	-0.0589515	-0.2630688
FAC_2	0.1826295	0.1223784	0.0827045	0.0979092	0.0437228

Eigenvectors

	11	12	13	14	15
FAC_1	0.3296719	0.2028505	0.0549809	-0.1309457	-0.3169474
FAC_2	-0.0137984	-0.0550292	-0.0811586	-0.1116958	-0.1446445

Eigenvectors

	16	17	18	19	20
FAC_1	0.3946226	0.2940454	0.1607963	-0.0150352	-0.1956892
FAC_2	-0.1351410	-0.1878072	-0.2402878	-0.2918104	-0.3365611

Eigenvectors

	21
FAC_1	-0.3885698
FAC_2	-0.3713109

Unrotated Factor Loadings

	FAC_1	FAC_2
V_1	-0.0543280	0.0480745
V_2	-0.1064314	0.1305408
V_3	-0.1797051	0.2332687
V_4	-0.2606939	0.3162974
V_5	-0.3395128	0.3479646
V_6	-0.4142059	0.3337099
V_7	-0.4877752	0.2871232
V_8	-0.5582785	0.2136800
V_9	-0.6270921	0.1253097
V_10	-0.7033861	0.0367309

```
                        .........................
                V_265   0.0024751 -0.0150408
                V_266   0.0024407 -0.0152983
                V_267   0.0025101 -0.0154560
                V_268   0.0026621 -0.0155474


          Rotated Factor Loadings: Varimax (Kaiser, 1958)

                            FAC_1      FAC_2
                V_1     -0.0543280  0.0480745
                V_2     -0.1064314  0.1305408
                V_3     -0.1797051  0.2332687
                V_4     -0.2606939  0.3162974
                V_5     -0.3395128  0.3479646
                V_6     -0.4142059  0.3337099
                V_7     -0.4877752  0.2871232
                V_8     -0.5582785  0.2136800
                V_9     -0.6270921  0.1253097
                V_10    -0.7033861  0.0367309
                .........................
                V_265   0.0024751 -0.0150408
                V_266   0.0024407 -0.0152983
                V_267   0.0025101 -0.0154560
                V_268   0.0026621 -0.0155474


         Orthogonal Transformation Matrix T

                            FAC_1      FAC_2
              FAC_1     1.0000000  0.0000000
              FAC_2     0.0000000  1.0000000
```

Standard Errors for Unrotated and Rotated PCA:

Only the correlation matrix is given. Therefore means are treated as zeros and standard deviations as ones.

```
print "Six School Subjects (Lawley and Maxwell, 1971, p.66)";
corr = [   1. ,
         .439   1. ,
         .410 .351   1. ,
         .288 .354 .164   1. ,
         .329 .320 .190 .595   1. ,
         .248 .329 .181 .470 .464  1. ];
corr = (tri2sym)corr;
```

```
nobs = 220;
```

The following output is similar for all following examples:

```
                    ****************************
                    Principal Component Analysis
                    ****************************

  Input Data. . . . . . . . . . . . . . . .Correlation Matrix
  Analysis of . . . . . . . . . . . . . . . Covariance Matrix
  Number of Items . . . . . . . . . . . . . . . . . . .     6
  Specified Number of Factors . . . . . . . . . . . .     2
  Number of Subjects. . . . . . . . . . . . . . . . .   220
  Version . . . . . . . . . . . . . . . .Standard Full EVD Code
  Rotation Method . . . . . . . . . . . . . . . . No Rotation


                    Name             Mean     Std Dev
                    ---------------------------------
                    V_1       0.000000000  1.00000000
                    V_2       0.000000000  1.00000000
                    V_3       0.000000000  1.00000000
                    V_4       0.000000000  1.00000000
                    V_5       0.000000000  1.00000000
                    V_6       0.000000000  1.00000000


                           Covariance Matrix

              V_1        V_2        V_3        V_4        V_5
  V_1   1.0000000
  V_2   0.4390000  1.0000000
  V_3   0.4100000  0.3510000  1.0000000
  V_4   0.2880000  0.3540000  0.1640000  1.0000000
  V_5   0.3290000  0.3200000  0.1900000  0.5950000  1.0000000
  V_6   0.2480000  0.3290000  0.1810000  0.4700000  0.4640000


                           Covariance Matrix

                              V_6
                        V_6   1.0000000
      Determinant =  2.37409534e-001 (Ln = -1.4380e+000)
```

```
Eigenvalues of Model Covariance Matrix:
Total = 6.0000      Average = 1.0000


                        1           2           3           4
     Eigenvalue  2.73288407  1.12977037  0.61517387  0.60122188
     Difference  1.60311371  0.51459650  0.01395199  0.07642497
     Proportion  0.45548068  0.18829506  0.10252898  0.10020365
     Cumulative  0.45548068  0.64377574  0.74630472  0.84650836


                        5           6
     Eigenvalue  0.52479691  0.39615290
     Difference  0.12864401
     Proportion  0.08746615  0.06602548
     Cumulative  0.93397452  1.00000000


                          Eigenvectors

               V_1         V_2         V_3         V_4         V_5
  FAC_1  -0.3979211 -0.4164293 -0.3129591 -0.4466126 -0.4499766
  FAC_2   0.4224739  0.2732052  0.5996225 -0.3885864 -0.3532281


                          Eigenvectors

                          V_6
                    FAC_1  -0.4103173
                    FAC_2  -0.3340032
```

1. Unstandardized Unrotated PCA:

```
    optn = [ "data"      "cor" ,
             "anal"      "cov" ,
             "nobs"        220 ,
             "vers"     "cev1" ,
             "nfac"          2 ,
             "cl"       "wald" ,
             "prin"          3 ];
    < gof,eval,flod > = pca(corr,optn);


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

```
                    Principal Component Analysis
                    ****************************

  Input Data. . . . . . . . . . . . . . . .Correlation Matrix
  Analysis of . . . . . . . . . . . . . . . Covariance Matrix
  Number of Items . . . . . . . . . . . . . .         6
  Specified Number of Factors . . . . . . . .         2
  Number of Subjects. . . . . . . . . . . . .       220
  Version . . . . . . . . . . . . . . .Standard Full EVD Code
  Rotation Method . . . . . . . . . . . . . . No Rotation


        Unrotated Factor Loadings with Standard Errors
        ----------------------------------------------

                            FAC_1                  FAC_2
      V_1    0.6578207  0.0793558  0.4490503  0.0895841
           [ 0.502286, 0.813355] [ 0.273469, 0.624632]

      V_2    0.6884175  0.0741050  0.2903916  0.1015619
           [ 0.543174, 0.833661] [ 0.091334, 0.489449]

      V_3    0.5173663  0.0919484  0.6373427  0.0918856
           [ 0.337151, 0.697582] [ 0.457250, 0.817435]

      V_4    0.7383147  0.0733037 -0.4130310  0.0732165
           [ 0.594642, 0.881987] [-0.556533,-0.269529]

      V_5    0.7438760  0.0720214 -0.3754484  0.0776596
           [ 0.602717, 0.885035] [-0.527658,-0.223238]

      V_6    0.6783135  0.0759136 -0.3550142  0.0966014
           [ 0.529525, 0.827101] [-0.544350,-0.165679]
```

2. Standardized Unrotated PCA:

```
optn = [ "data"      "cor" ,
         "anal"      "cor" ,
         "nobs"        220 ,
         "vers"     "cev1" ,
         "nfac"          2 ,
         "cl"       "wald" ,
         "frot"     "none" ,
         "prin"          3 ];
< gof, est > = pca(corr,optn);
```

```
                    ****************************
                    Principal Component Analysis
                    ****************************

Input Data. . . . . . . . . . . . . . .Correlation Matrix
Analysis of . . . . . . . . . . . . . .Correlation Matrix
Number of Items . . . . . . . . . . . . . . . . .      6
Specified Number of Factors . . . . . . . . . .      2
Number of Subjects. . . . . . . . . . . . . . .    220
Version . . . . . . . . . . . . . .Standard Full EVD Code
Rotation Method . . . . . . . . . . . . . . . No Rotation


       Unrotated Factor Loadings with Standard Errors
       ----------------------------------------------

                          FAC_1                 FAC_2
     V_1   0.6578207  0.0463014  0.4490503  0.0735708
          [ 0.567072, 0.748570] [ 0.304854, 0.593246]

     V_2   0.6884175  0.0411787  0.2903916  0.0860066
          [ 0.607709, 0.769126] [ 0.121822, 0.458961]

     V_3   0.5173663  0.0664515  0.6373427  0.0713588
          [ 0.387124, 0.647609] [ 0.497482, 0.777203]

     V_4   0.7383147  0.0341763 -0.4130310  0.0613606
          [ 0.671330, 0.805299] [-0.533296,-0.292766]

     V_5   0.7438760  0.0332094 -0.3754484  0.0659694
          [ 0.678787, 0.808965] [-0.504746,-0.246151]

     V_6   0.6783135  0.0427755 -0.3550142  0.0830693
          [ 0.594475, 0.762152] [-0.517827,-0.192201]
```

3. Unstandardized Normed Orthogonally Rotated PCA:

```
optn = [ "data"      "cor" ,
         "anal"      "cov" ,
         "nobs"        220 ,
         "vers"     "cev1" ,
         "nfac"          2 ,
         "cl"       "wald" ,
         "rvers"    "comm" ,
```

```
        "frot"   "varmax" ,
        "fnorm"          ,
        "prin"         3 ];
< gof, est > = pca(corr,optn);
```

```
                ****************************
                Principal Component Analysis
                ****************************

  Input Data. . . . . . . . . . . . . . . .Correlation Matrix
  Analysis of . . . . . . . . . . . . . . . Covariance Matrix
  Number of Items . . . . . . . . . . . . . .         6
  Specified Number of Factors . . . . . . . .         2
  Number of Subjects. . . . . . . . . . . . .       220
  Version . . . . . . . . . . . . . .Standard Full EVD Code
  Rotation Method . . . . . . . . . .Varimax (Kaiser, 1958)
  Normed Rotation Method. . . . . . . . . . . . . 1.0000000
```

Unrotated Factor Loadings

	FAC_1	FAC_2
V_1	0.6578207	0.4490503
V_2	0.6884175	0.2903916
V_3	0.5173663	0.6373427
V_4	0.7383147	-0.4130310
V_5	0.7438760	-0.3754484
V_6	0.6783135	-0.3550142

Rotated Factor Loadings with Standard Errors

	FAC_1		FAC_2	
V_1	0.2253107	0.0734485	0.7639433	0.0790763
	[0.081354, 0.369267]		[0.608957, 0.918930]	
V_2	0.3493867	0.0923938	0.6604354	0.0928254
	[0.168298, 0.530475]		[0.478501, 0.842370]	
V_3	-0.0025887	0.0627659	0.8208940	0.0816661
	[-0.125608, 0.120430]		[0.660831, 0.980957]	
V_4	0.8330754	0.0642808	0.1472704	0.0625851
	[0.707087, 0.959063]		[0.024606, 0.269935]	

```
V_5   0.8136040  0.0662075  0.1798932  0.0641233
     [ 0.683840, 0.943368] [ 0.054214, 0.305573]

V_6   0.7499035  0.0786806  0.1542368  0.0791837
     [ 0.595692, 0.904115] [-9.6e-004, 0.309434]
```

4. Standardized Unnormed Orthogonally Rotated PCA:

```
optn = [ "data"      "cor" ,
         "anal"      "cor" ,
         "nobs"        220 ,
         "vers"     "cev1" ,
         "nfac"          2 ,
         "cl"       "wald" ,
         "rvers"    "comm" ,
         "frot"   "varmax" ,
         "prin"          3 ];
< gof, est > = pca(corr,optn);
```

```
                ****************************
                Principal Component Analysis
                ****************************

  Input Data. . . . . . . . . . . . . . . .Correlation Matrix
  Analysis of . . . . . . . . . . . . . . .Correlation Matrix
  Number of Items . . . . . . . . . . . . . . . . . .        6
  Specified Number of Factors . . . . . . . . . . .        2
  Number of Subjects. . . . . . . . . . . . . . . .      220
  Version . . . . . . . . . . . . . . . .Standard Full EVD Code
  Rotation Method . . . . . . . . . . . .Varimax (Kaiser, 1958)
  Unnormed Rotation Method. . . . . . . . . . . . . 1.0000000


                  Unrotated Factor Loadings

                          FAC_1      FAC_2
                  V_1   0.6578207  0.4490503
                  V_2   0.6884175  0.2903916
                  V_3   0.5173663  0.6373427
                  V_4   0.7383147 -0.4130310
                  V_5   0.7438760 -0.3754484
                  V_6   0.6783135 -0.3550142


            Rotated Factor Loadings with Standard Errors
            --------------------------------------------
```

```
                    FAC_1                FAC_2
V_1   0.2386811  0.0759140  0.7598721  0.0451430
     [ 0.089892, 0.387470] [ 0.671393, 0.848351]

V_2   0.3609217  0.0907056  0.6542029  0.0718067
     [ 0.183142, 0.538701] [ 0.513464, 0.794941]

V_3   0.0118162  0.0585867  0.8208130  0.0340498
     [-0.103012, 0.126644] [ 0.754077, 0.887549]

V_4   0.8355314  0.0251807  0.1326295  0.0597901
     [ 0.786178, 0.884885] [ 0.015443, 0.249816]

V_5   0.8166354  0.0292857  0.1655889  0.0616674
     [ 0.759237, 0.874034] [ 0.044723, 0.286455]

V_6   0.7524945  0.0435023  0.1410543  0.0827566
     [ 0.667232, 0.837757] [-0.021146, 0.303254]
```

5. Unstandardized Normed Oblique Rotated PCA:

```
optn = [ "data"      "cor" ,
         "anal"      "cov" ,
         "nobs"        220 ,
         "vers"     "ogas" ,
         "nfac"          2 ,
         "cl"       "wald" ,
         "rvers"    "comm" ,
         "frot"   "quamin" ,
         "fnorm"           ,
         "prin"          3 ];
< gof, est > = pca(corr,optn);


                ***************************
                Principal Component Analysis
                ***************************

  Input Data. . . . . .Correlation Matrix
  Analysis of . . . . . Covariance Matrix
  Number of Items . . . . . 6
  Specified Number of Factors . . . . . 2
  Number of Subjects. . . . . 220
  Version . . . . . .Standard Full EVD Code
  Rotation Method . . . . . Quartimin (Carroll, 1953)
```

Normed Rotation Method. 0.0000000

Unrotated Factor Loadings

	FAC_1	FAC_2
V_1	0.6578207	0.4490503
V_2	0.6884175	0.2903916
V_3	0.5173663	0.6373427
V_4	0.7383147	-0.4130310
V_5	0.7438760	-0.3754484
V_6	0.6783135	-0.3550142

Orthomax Prerotated Loadings: VARIMAX

	FAC_1	FAC_2
V_1	0.2253107	0.7639433
V_2	0.3493867	0.6604354
V_3	-0.0025887	0.8208940
V_4	0.8330754	0.1472704
V_5	0.8136040	0.1798932
V_6	0.7499035	0.1542368

Rotated Factor Loadings with Standard Errors

	FAC_1		FAC_2	
V_1	0.0871917	0.0848962	0.7604488	0.0894415
	[-0.079202, 0.253585]		[0.585147, 0.935751]	
V_2	0.2386581	0.1126435	0.6261073	0.1088639
	[0.017881, 0.459435]		[0.412738, 0.839477]	
V_3	-0.1647378	0.0534282	0.8666891	0.0901401
	[-0.269455,-0.060020]		[0.690018, 1.043360]	
V_4	0.8507661	0.0715458	-0.0133039	0.0644292
	[0.710539, 0.990993]		[-0.139583, 0.112975]	
V_5	0.8237643	0.0738101	0.0250612	0.0676976
	[0.679099, 0.968429]		[-0.107624, 0.157746]	
V_6	0.7615520	0.0892199	0.0108893	0.0845470
	[0.586684, 0.936420]		[-0.154820, 0.176598]	

```
        Factor Correlations with Standard Errors
        ----------------------------------------

                          FAC_1                 FAC_2
FAC_1   1.0000000  0.0000000  0.3656592  0.0542791
       [ 1.000000, 1.000000] [ 0.259274, 0.472044]

FAC_2   0.3656592  0.0542791  1.0000000  0.0000000
       [ 0.259274, 0.472044] [ 1.000000, 1.000000]
```

6. Standardized Unnormed Oblique Rotated PCA:

```
optn = [ "data"      "cor" ,
         "anal"      "cor" ,
         "nobs"        220 ,
         "vers"     "cev1" ,
         "comp"            ,
         "nfac"          2 ,
         "tech"   "nrridg" ,
         "cl"       "wald" ,
         "rvers"    "comm" ,
         "frot"   "quamin" ,
         "prin"          3 ];
< gof, est > = pca(corr,optn);


                ****************************
                Principal Component Analysis
                ****************************

  Input Data. . . . . .Correlation Matrix
  Analysis of . . . . . .Correlation Matrix
  Number of Items . . . . . 6
  Specified Number of Factors . . . . . 2
  Number of Subjects. . . . . 220
  Version . . . . . .Standard Full EVD Code
  Rotation Method . . . . . Quartimin (Carroll, 1953)
  Unnormed Rotation Method. . . . . 0.0000000


                Unrotated Factor Loadings

                     FAC_1      FAC_2
                V_1  0.6578207  0.4490503
                V_2  0.6884175  0.2903916
                V_3  0.5173663  0.6373427
```

```
V_4   0.7383147 -0.4130310
V_5   0.7438760 -0.3754484
V_6   0.6783135 -0.3550142
```

Orthomax Prerotated Loadings: VARIMAX

```
          FAC_1      FAC_2
V_1   0.2253107  0.7639433
V_2   0.3493867  0.6604354
V_3  -0.0025887  0.8208940
V_4   0.8330754  0.1472704
V_5   0.8136040  0.1798932
V_6   0.7499035  0.1542368
```

Rotated Factor Loadings with Standard Errors

```
                    FAC_1                 FAC_2
V_1   0.1070191  0.0961876  0.7532025  0.0606586
     [-0.081505, 0.295543] [ 0.634314, 0.872091]

V_2   0.2551183  0.1190878  0.6197485  0.0923557
     [ 0.021710, 0.488526] [ 0.438735, 0.800762]

V_3  -0.1423536  0.0356829  0.8590523  0.0348038
     [-0.212291,-0.072416] [ 0.790838, 0.927267]

V_4   0.8511095  0.0324862 -0.0151831  0.0558954
     [ 0.787438, 0.914781] [-0.124736, 0.094370]

V_5   0.8250821  0.0378689  0.0228900  0.0617596
     [ 0.750860, 0.899304] [-0.098157, 0.143937]

V_6   0.7624512  0.0574010  0.0089956  0.0916217
     [ 0.649947, 0.874955] [-0.170580, 0.188571]
```

Factor Correlations with Standard Errors

```
                     FAC_1                 FAC_2
FAC_1   1.0000000  0.0000000  0.3449275  0.0622212
       [ 1.000000, 1.000000] [ 0.222976, 0.466879]

FAC_2   0.3449275  0.0622212  1.0000000  0.0000000
```

```
[ 0.222976, 0.466879] [ 1.000000, 1.000000]
```

3 New Developments

3.1 Function `centroid`

<u,v,d> = centroid(a,k<optn>)

Purpose: The `centroid` function implements both the new Chu & Funderlik (2002) and the classical Thurstone (1931) *centroid* method for factor analysis. The model is

$$\mathbf{A} = \mathbf{V}\mathbf{B}^T \quad with \quad \mathbf{V}^T\mathbf{V} = I_k$$

where $\mathbf{A}$ is the $m \times n$ input data matrix and is the $m \times k$ factor score matrix and $\mathbf{B}$ is the $n \times k$ factor loading matrix. The algorithm also computes a $k \times k$ diagonal matrix $\mathbf{D}$ of centroid values which are equivalent to the singular values of a corresponding model but are here contained in the column norms of matrix $\mathbf{B}$.

Input: **a** is a given $n \times k$ (usually sparse) matrix.

k is the number of factors, i.e. the column dimension of matrices **U** and **H**.

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
`"meth"`	string	algorithm used for estimation
	"chufu"	method by Chu & Funderlik (2002)
	"class"	the classic method
`"print"`	int	amount of printed output (=0: no output)

Output:

Restrictions: 1. The $m \times n$ input matrix **A** should have neither missing value nor complex or string data.

2. The number of factors k should be not larger than the smaller of the dimensions $\min(m, n)$ of the input matrix **A**.

Relationships: factor(), pca(), svd(), nnmf()

Examples: Here we use the same data set as is used for the non-negative matrix factorization:

```
v =  [ 4  1  1,   7  1  1,   3  1  1,   5  1  1,   1  1  1,
       6  1  1,   2  1  2,   7  2  1,   9  2  1,  10  2  1,
```

```
 11  2  1,   8  2  1,   2  2  2,  13  3  1,  12  3  1,
 14  3  1,  11  3  1,   9  4  1,  15  4  1,   8  4  2,
  2  4  1,  16  4  1,   7  5  1,   9  5  1,  17  5  1,
 18  5  1,  10  5  1,  19  5  1,   2  5  2,  21  6  1,
  7  6  1,  12  6  1,  23  6  1,   3  6  1,  14  6  1,
 24  6  1,  22  6  1,   2  6  1,  20  6  1,   7  7  1,
  9  7  1,  17  7  1,  18  7  1,  15  7  1,  19  7  1,
  8  7  1,   2  7  1,   5  8  1,  15  8  1,  25  8  1,
 26  8  1,  28  8  1,   2  8  2,  27  8  1,  29  9  1,
 25  9  1,  26  9  1,   2  9  1 ];
```

```
nr = 29; nc = 9; nf = 2;
a = spmat(nr,nc,v[,1],v[,2],v[,3]);

rnam = [ "Row1" : "Row29" ];
cnam = [ "Col1" : "Col9" ];
a = rname(a,rnam);
a = cname(a,cnam);
print "A=", a; /* attrib(a); */
```

1. First we show the result of the Chu & Funderlik (2002) algorithm:

```
optn = [   2 ,      /* ipri */
           1 ];     /* imet=chufun */
/* options debug="ollss*=4 centr*=4"; */
< v,b,d,z > = centroid(a,nf,optn);

print "Residual=",ssq(a - v * b‘);
print "Z*Z=", z‘ * z;
print "V*V=", v‘ * v;
print "B*B=", b‘ * b;
print "Matrix D:", d;

print "Matrix B:", b;
print "Matrix V:", v;
print "Matrix Z:", z;
```

```
Residual= 29.354
```

Z*Z=

S	1	2
1	9.00000	
2	-1.00000	9.00000

V*V=

S	1	2
1	1.00000	
2	4e-017	1.00000

B*B=

S	1	2
1	33.5019	
2	2.3547	10.1437

Matrix D:

D	1	2
1	9.20690	0
2	0	2.94305

Matrix B:

	1	2
1	2.20316	-0.95595
2	2.50916	0.84162
3	0.42839	-0.60081
4	1.71357	1.27703
5	2.50916	1.05811
6	1.71357	-1.32082
7	2.08077	1.44244
8	2.14197	-1.05568
9	1.04038	-0.68595

```
Matrix V:
   |         1         2
-------------------------
 1 |   0.06120  -0.11676
 2 |   0.73439  -0.10216
 3 |   0.12240  -0.23351
 4 |   0.06120  -0.11676
 5 |   0.12240  -0.23351
 6 |   0.06120  -0.11676
 7 |   0.30600   0.06568
 8 |   0.24480   0.39892
 9 |   0.24480   0.39892
10 |   0.12240   0.19946
11 |   0.12240  -0.01703
12 |   0.12240  -0.23351
13 |   0.06120  -0.11676
14 |   0.12240  -0.23351
15 |   0.18360   0.08270
16 |   0.06120   0.09973
17 |   0.12240   0.19946
18 |   0.12240   0.19946
19 |   0.12240   0.19946
20 |   0.06120  -0.11676
21 |   0.06120  -0.11676
22 |   0.06120  -0.11676
23 |   0.06120  -0.11676
24 |   0.06120  -0.11676
25 |   0.12240  -0.23351
26 |   0.12240  -0.23351
27 |   0.06120  -0.11676
28 |   0.06120  -0.11676
29 |   0.06120  -0.11676
```

```
Matrix Z:
   |         1         2
-------------------------
 1 |   1.00000  -1.00000
 2 |   1.00000   1.00000
 3 |   1.00000  -1.00000
 4 |   1.00000   1.00000
 5 |   1.00000   1.00000
 6 |   1.00000  -1.00000
 7 |   1.00000   1.00000
 8 |   1.00000  -1.00000
 9 |   1.00000  -1.00000
```

2. Now we run the classic algorithm:

```
optn = [   2 ,      /* ipri */
           2 ];     /* imet=classic */
/* options debug="ollss*=4 centr*=4"; */
< v,b,d,z > = centroid(a,nf,optn);

print "Residual=",ssq(a - v * b‘);
print "Z*Z=", z‘ * z;
print "V*V=", v‘ * v;
print "B*B=", b‘ * b;
print "Matrix D:", d;
```

```
  print "Matrix B:", b;
  print "Matrix V:", v;
  print "Matrix Z:", z;
```

```
Residual= 29.354
```

```
Z*Z=                                    B*B=
 S |         1         2                 S |         1         2
-------------------------               -------------------------
 1 |   9.00000                           1 |   33.5019
 2 |  -1.00000   9.00000                 2 |    2.3547   10.1437

V*V=                                    Matrix D:
 S |         1         2                 D |         1         2
-------------------------               -------------------------
 1 |   1.00000                           1 |   9.20690         0
 2 |    1e-016   1.00000                 2 |         0   2.94305
```

Also the remaining results are the same as for the Chu & Funderlik (2002) method.

3.2 Function `generead`

a = generead(fpath,resp,sep<,optn<,scod>>)

Purpose: The `generead` function reads comma or otherwise separated ASCI data sets into a numeric object. The object can be transposed and the separator between numbers of a row can be specified. The number of rows and columns must not be known before calling this function. A nominal or ordinal (CLASS) string response variable can be mapped into a integer row or column. This function also permits the input of a special file type where strings and numeric values are arranged either pairwise (ftyp=1) or in blocks of equal size (ftyp=2). Such kind of data sets are common for micro array data.

Input: **fpath** This argument must be a single string argument specifying the file path to the input data set.

resp This argument should be missing if there is no response variable. Otherwise it must be a vector containing strings defining each level of a response variable.

sep This argument must be a string argument specifying the separator which is used in the file between numbers of the same row. The end of a row and begin of a new row is defined by a new line without a separator.

optn This argument must be a numeric scalar or vector specifying the following options:

1. a nonmissing unequal zero input specifies that the transpose of the data set is returned
2. a zero or missing specifies that the data contains no response variable; a positive integer specifies the column or row (transposed) of a response variable;
3. a nonmissing unequal zero input specifies that the file contains row names
4. a nonmissing unequal zero input specifies that the file contains column names
5. currently unused
6. for special file types only: ftyp=0: file contains only expression data (default) ftyp=1: (str,val),(str,val),...,(str,val),resp ftyp=2: (str,...,str),(val,...,val),resp
7. for special file types only: cinc: =1: valid only for ftyp != 0: return only expression data, no strings

scod for special file types only: This argument must be a vector of *ncod* strings; default is *ncod* = 3: "A": Absent, "M": Marginal, "P": Present

Output: The result is a $m \times n$ matrix object **A** containing the data of the input file. If the data set contains row and/or column names they are attached to the result object **A**.

Restrictions:
1. String data are only permitted for: row and column names, the response, and for the special file type
2. Missing values are permitted.

Relationships:

Examples:
1. Myeloma Data set: Affymetrix micro array data:
   - only expression value: nobs=105, nvar=7009
   - with A,M,P char variable: nobs=105, nvar=28033

   The first reading does not include the char vars and returns an object with only 7009 columns:

```
resp = [ "normal" ,
         "myeloma" ];
optn = cons(8,1,.);
optn[6] = 2;   /* ftyp */
optn[7] = 1;   /* cinc */
myelo = generead("..\\tdata\\myeloma.dat",resp,",",optn);
```

```
print "nc=", nc = ncol(myelo),
      "nr=", nr = nrow(myelo);
attrib(myelo);
obj2fil(myelo,"..\\save\\myelo");
```

2. The second reading of the same data also includes the char vars and returns an object with 28033 columns:

```
resp = [ "normal" ,
         "myeloma" ];
optn = cons(8,1,.);
optn[6] = 2;   /* ftyp */
optn[7] = 0;   /* cinc */
myelo = generead("..\\tdata\\myeloma.dat",resp,",",optn);


print "nc=", nc = ncol(myelo),
      "nr=", nr = nrow(myelo);
attrib(myelo);
obj2fil(myelo,"..\\save\\myelo2");
```

3. Stan's Data Set:

```
stan2 = "C:\\tdata2\\stan\\stan2.dat";
optn = cons(8,1,.);
optn[1] = 0;   /* itra  */
optn[2] = 0;   /* iresp */
optn[3] = 1;   /* irnam */
optn[4] = 1;   /* icnam */
stan2 = generead(stan2,.,",",optn);


print "nc=", nctst = ncol(stan2),
      "nr=", nrtst = nrow(stan2);
/* print "First three Vars", stan2[, 1:3 ];  */
print "Colnames=", cnam = cname(stan2);
print "Rownames=", rnam = rname(stan2);
/* create object file save/stan2 */
attrib(stan2);
obj2fil(stan2,"..\\save\\stan2");
```

4. Roche 18 kinds of Leukemia Affymetrix Microarray Data:
   - Train: nobs= 719, nvar= 29501;
   - Test : nobs= 190
   - Response: multinomial with 18 categories

```
ftrn = "C:\\tdata2\\roche\\roche_trn.dat";
optn = cons(8,1,.);
optn[1] = 1;   /* itra  */
optn[2] = 1;   /* iresp */
optn[3] = 1;   /* irnam */
optn[4] = 0;   /* icnam */
roche_trn = generead(ftrn,.,",",optn);


print " Train nc=", nctrn = ncol(roche_trn),
      " nr=", nrtrn = nrow(roche_trn);
attrib(roche_trn);
print "First 5 obs for Training", roche_trn[1:5,29500:29502];

obj2fil(roche_trn,"..\\save\\roche_trn");
/* print "Response in Train", roche_trn[,29502]; */
```

5. Breast Cancer Genomic Data: Laura J. van't Veer:
   - Letters to Nature, Jan. 2002
   - New England J. of Medicine, Dec. 2002
   - Train: nobs= 98, nvar= 24191
   - Response: ordinal, make binary with cutoff ¡= 60

```
ftrn98 = "C:\\tdata2\\vanVeer\\van_Veer98.dat";
optn = cons(8,1,.);
optn[1] = 1;   /* itra  */
optn[2] = 1;   /* iresp */
optn[3] = 1;   /* irnam */
optn[4] = 0;   /* icnam */
vanVeer98 = generead(ftrn98,.,",",optn);


print "nc=", nctst = ncol(vanVeer98),
      "nr=", nrtst = nrow(vanVeer98);
print "First three Vars", vanVeer98[, 1:3 ];
print "Ordinal Response", vanVeer98[, 24189 ];
attrib(vanVeer98);
cnam = cname(vanVeer98);
```

```
print "First three Names: vanVeer98=", cnam[ 1:3 ];
print "Last three Names: vanVeer98=", cnam[ 24187:24189 ];
obj2fil(vanVeer98,"..\\save\\vanVeer98");
print "Ordinal Response", vanVeer98[ ,24189 ];
```

3.3 Function `histogrm`

hist = histogrm(a,k,<,optn>)

Purpose: The `histogrm` function computes one k histogram for each of the n columns of an m *timesn* data matrix `A`. The options vector permits specifying an enclosing range for all n histograms or the setting of lower or upper bounds for each histogram.

Input: **a** The first input argument specifies a $m \times n$ data matrix.

k The integer k specifies the number of intervals i.e. $k = 4$ for four intervals.

optn A numeric scalar or vector specifies the following options:

1. specifies the print level;
2. an integer unequal zero specifies that the minimum and maximum of all columns is used for the lower and upper range of each histogram;
3. a real number specifying a common lower range;
4. a real number specifying a common upper range.

Output: The result is a $n \times k$ matrix containing the frequencies of observations (rows) in the k intervals for each of the n columns.

Restrictions:
1. Missing values in input data `a` are not counted for quantiles.
2. No complex or string data can be used.

Relationships: univar(), quantile()

Examples: 1. Heart data, D.M. Hawkins (1994)

```
a= [ 1 42.8 40.0 37,
     2 63.5 93.5 50,
     3 37.5 35.5 34,
     4 39.5 30.0 36,
     5 45.5 52.0 43,
     6 38.5 17.0 28,
     7 43.0 38.5 37,
```

```
 8 22.5  8.5 20,
 9 37.0 33.0 34,
10 23.5  9.5 30,
11 33.0 21.0 38,
12 58.0 79.0 47 ];


optn = [ 2 ];
hist = histogrm(a,4,optn);
print "Histogram(same=0)=", hist;
```

The rows in the histogram table must add to the number of observations (rows) in the data matrix.

```
          Histogram
          *********

    Dense Matrix (3 by 4)

  |       1      2      3      4
--------------------------------
1 |       2      7      1      2
2 |       4      5      1      2
3 |       1      4      4      3


optn = [ 2 ,    /* ipri */
         1 ];   /* same */
hist = histogrm(aa,4,optn);
print "Histogram(same=1)=", hist;


          Histogram
          *********

    Dense Matrix (3 by 4)

  |       1      2      3      4
--------------------------------
1 |       2      8      2      0
2 |       4      5      1      2
3 |       2     10      0      0
```

3.4 Function `impute`

<xful,scal> = impute(xmis,sopt<optn<,class<,bounds>>>)

Purpose: The `impute` function implements a variety of algorithms for the imputation of missing values. It is assumed that the N rows of the $N \times n$ data set `xmis` corresopond to observations (cases) and its n columns correspond to variables. Note, that the alternating regressions methods implemented with the `svd` function permit missing value patterns in the input data matrix and therefore can also be used as an alternative for `impute`. The EM estimation of mean and covariance matrix for data with missing values is implemented with the `emcov` function.

The `impute` function implements two groups of algorithms:

1. A set of algorithms that is noniterative and fills the missing values with initial values depending only on the information of given data.
2. A set of iterative methods that starts with initial values created by a method of the first set of algorithms and improving the imputed values in an EM (Expectation Maximization) like way until (hopefully) convergence is reached.

The following methods for imputation are currently available:

"none" can only be used for iterative method and specifies that only initial imputaton is performed.

"scalar" replaces all missing values with a constant which can be specified with the `optn` argument. That could be useful for creating a data set for input with another program which has a different standard for coding missing values than CMAT, e.g. SPSS codes missing values as "99".

"randuni" replaces all missing values with a uniform random value in the columnwise range $[xmin, xmax]$ (useful for numeric and CLASS columns)

"randnrm" replaces all missing values with a randomly normal (ν, σ) generated value where ν and σ are the mean and standard deviation for the nonmissing values in each column.

"randtab" replaces all missing values with random values with probabilities conform to the tabled values of the levels of a CLASS variable (preferred for CLASS columns)

"colmean" replaces all missing values of

- numeric variables: with the mean
- categorical variables: with the median

of the nonmissing values in each column.

"mindist" implements the missing values of each observation by using the specified values of an the observation with minimum distance. (useful for numeric and CLASS columns)

"knearn" implements the K nearest neighbor method: for each observation (case) with missing values the K nearest neighbor observations are found and the missing values of

- numeric variables: are replaced with the mean
- categorical variables: are replaced with the median

of the nonmissing values in each column.

"linreg" implements a linear regression algorithm for predicting the missing values of one column from all of the remaining columns based on the common linear least squares regression model. Preferred for $N >> n$

"frwreg" implements a linear regression algorithm for predicting the missing values of one column from a forward selection within the subset of the remaining $n - 1$ columns based on the common linear least squares regression model. Preferred for $N << n$.

"logreg" implements logistic regression algorithm for predicting the missing values of one column from a forward selection of a subset of the remaining columns based on the common binary response logistic regression model. Preferred for CLASS columns.

"simpls" implements the linear SIMPLS (partial least squares) algorithm for predicting the missing values of one column from the given information of the remaining columns.

"krnpls" implements the linear kernel PLS (partial least squares) algorithm for predicting the missing values of one column from the given information of the remaining columns.

The `"linreg"` and `"frwreg"` differ in the choice of predictor variables. Whereas the `"linreg"` method uses always all $n - 1$ remaining columns as predictors the `"frwreg"` method is using only a subset of columns selected stepwise based on the largest partial correlations. The stepwise selection is terminated when the largest partial R^2 value of remaining unselected variables becomes smaller than a specified threshold. In general, the `"linreg"` method may have advantages for $N >> n$ whereas the `"frwreg"` method should be used for $N << n$.

For CLASS variables (binary, nominal, or ordinal scaled) currently only the `"scalar"`, `"randuni"`, `randtab"`, `"logreg"`, and `"mindist"` methods can be used. From those `"scalar"`, `"randuni"`, and `"mindist"` can be used for both, numeric and CLASS data columns.

Among those the following methods can only be used for generating initial values:

"scalar" for numeric and CLASS

"randuni" for numeric and CLASS

"randnrm" preferred for numeric column

"randtab" preferred for CLASS column

"colmean" for numeric column with mean for CLASS with median

If one of those methods is specified as final method, it is ignored and the initial values are not modified.

The following method names can be used for both group specification even though that they are implemented in very different ways for each of the two purposes:

"mindist" for numeric and CLASS

"knearn" for numeric (with mean) and CLASS (with median)

"linreg" for numeric and CLASS

"frwreg" for numeric and CLASS

"logreg" preferred for CLASS column

"simpls" preferred for numeric column

"krnpls" preferred for numeric column

The use of these algorithms for initial estimates must not always be feasible since it assumes a reasonable large subset of observations and variables without missing values for modeling. When the specified number of components for the `"simpls"` and the `"krnpls"` algorithms is larger than the number of variables in a feasible subset for initial estimation using **only** given information, it will be reduced for the specific subproblem.

Input: xmis is a given $N \times n$ matrix containing missing values.

sopt this is either

1. a string scalar denoting the initial and final imputation method for all columns.
2. a vector of n strings denoting the initial and final imputation method for each column.
3. a $n\times$ matrix of strings denoting the initial (first column) and final (second column) imputation method for all n columns.

The reason for columnwise specification of imputation methods is that the data columns may be scaled in different ways. E.g. for nominal or ordinal scaled *CLASS* variables data columns other imnputation methods can be suitable than for numeric (interval or ratio) scaled variables. The following imputation methods are available:

"none"

"scalar"

"randuni"
"randnrm"
"randtab"
"colmean"
"mindist"
"knearn"
"linreg"
"frwreg"
"logreg"
"simpls"
"krnpls"

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.

class (optional) This optional argument should be an integer scalar or vector of integer scalars naming the number of columns which are considered categorical (nominal scaled) variables.

bounds (optional) specifies a n vector with lower bounds or $n \times 2$ matrix with lower (first column) and upper (second column) boundary constraints for the imputed values of each column. Specifying a missing value for a lower or upper bound is equivalent to the extrem smallest and largest double precision values. Note, that the `"datab"` option enforces bounds on the imputed values as defined by the lower and upper bound of each nonmissing values in every column of the data.

Option	Second Column	Meaning
"cent"		the imputation algorithm works with centerd (w.r.t. mean) data
"const"	real	only for "scalar" imputation: the scalar value for imputation; default is 0;
"datab"		the imputed values must be within the bounds defined by the nonmissing data of column
"dist"	string	only for the "mindist" and "knearn" methods: the distance formula
	"l_2"	Euclidean distances
	"l_1"	City blocl distances
	"l_i"	maximum metric distances
"knn"	int	only for the "knearn" method: the number K of nearest neighbors chosen for averaging; default is $K = MIN(10, N/10)$
"maxit"	int	maximum number of iterations for the iterative imputation methods; default is 100
"memthr"	int	threshold for memory
"mincol"	int	only when using "linreg", "simpls", "krnpls" as initial methods: default is 2
"nfact"	int	only for the "simpls" and "krnpls" methods: the number of factors (latent vars). default is $MAX(1, n/2)$
"print"	int	specifies amount of printed output; default is 2;
"phist"	int	specifies history output for iterative methods;
"pinit"	int	specifies some output for initial values
"ppatt"	int	specifies some output of missing patterns
"ridg"	real	ridge value for "linreg" when there are more variables than observations; default is $1.e-6$
"rowper"	real	only when using "linreg", "simpls", "krnpls" as initial methods: default is 0.6
"scal"		the imputation algorithm works with scaled (w.r.t. standard deviation) data; default is 0
"start"	string	specifies the initialization method for the iterative imputation methods; default is "randnrm"
	"scalar"	see sopt argument
	"randuni"	see sopt argument
	"randnrm"	see sopt argument
	"colmean"	see sopt argument
	"mindist"	see sopt argument
	"knearn"	see sopt argument
	"linreg"	see sopt argument
	"frwreg"	see sopt argument
	"logreg"	see sopt argument
	"simpls"	see sopt argument
	"kernpls"	see sopt argument
"seed"	int	seed value for random generator; default is time of day;
"switch"	real	switching from dense to sparse pattern coding; default is 0.1
"tol"	real	termination tolerance for iteration process; default is $1.e-3$;

Output: **xful** is the $N \times n$ matrix which is the same as the input data matrix except that the missing values are now replaced by nonmissing numerical values.

scal is a vector with some information about the computaton.

Restrictions: 1. The `impute` function terminates with missing values when one or more entire columns of the input data matrix are missing.

2. The iterative algorithms may not converge or only slowly if there are many many or specific pattern of missing values. Centering and scaling (e.g. by the `"cent"` and `"scal"` options) of the data usually improves the convergence behavior.

Relationships: emcov(), svd()

Examples: 1. Heart Data: Weisberg (1980, p. 218):

```
heart = [ 1 42.8 40.0 37 ,
          2 63.5 93.5 50 ,
          3 37.5 35.5 34 ,
          4 39.5 30.0 36 ,
          5 45.5 52.0 43 ,
          6 38.5 17.0 28 ,
          7 43.0 38.5 37 ,
          8 22.5  8.5 20 ,
          9 37.0 33.0 34 ,
         10 23.5  9.5 30 ,
         11 33.0 21.0 38 ,
         12 58.0 79.0 47 ];
data = heart[,2:4];


/* some missing values are outliers:
   [2,1] and [2,3] are column maxima */
a = [ 1 42.8 40.0 37 ,
      2   .  93.5  . ,   /* 2 63.5 93.5 50 , */
      3 37.5 35.5 34 ,
      4 39.5   .  36 ,   /* 4 39.5 30.0 36 , */
      5 45.5 52.0 43 ,
      6 38.5 17.0 28 ,
      7   .  38.5 37 ,   /* 7 43.0 38.5 37 , */
      8 22.5  8.5 20 ,
      9 37.0 33.0 34 ,
     10 23.5  9.5  . ,   /* 10 23.5  9.5 30 , */
     11 33.0 21.0 38 ,
     12 58.0 79.0 47 ];
```

```
b = a[,2:4];
cnam = [ "v1":"v3" ];
b = cname(b,cnam);    /* print b; */


ubc = data[<>,];
lbc = data[><,]; /* print "bounds=", lbc, ubc; */
bounds = lbc' -> ubc'; print bounds;


/* KNEARN imputation and no Iteration:
   bounds does not enforce any constraints */
 optn = [ "print"            3 ,
          "ppatt"            2 ,
          "cent"               ,
          "scal"               ,
          "start"     "knearn" ,
          "knn"              2 ,
          "pinit"            2 ,
          "seed"           123 ,
          "tol"          1.e-3 ,
          "maxit"           30 ];
 < xful,scal > = impute(b,"none",optn,.,bounds);
```

```
   Number Observations . . . . . .          12
   Number Variables. . . . . .              3
   Number Missing Values . . . . . .        5
   Percentage of Missing Values. . . . . .  13.89
   Seed for Random Generator . . . . . .    123
   Maximum Number Iterations . . . . . .    30
   Termination Tolerance . . . . . .  0.00100000
   Nearest Neighbour K . . . . . .          2
```

```
            ************************************
            Column Summary for Imputation Method
            ************************************

Col Variable  Nmiss       Mean     StdDev   LowRange   UppRange
  1       v1      2  37.780000  10.317493  22.500000  58.000000
  2       v2      1  38.863636  27.197510  8.5000000  93.500000
  3       v3      2  35.400000  7.4565408  20.000000  47.000000

            ************************************
            Initial and Final Imputation Methods
            ************************************
```

```
 Column   Variable   Initial     Final    LowBound    UppBound
      1         v1    KNEARN      NONE  22.5000000  63.5000000
      2         v2    KNEARN      NONE  8.50000000  93.5000000
      3         v3    KNEARN      NONE  20.0000000  50.0000000

                  ***************************
                  Pattern of 5 Missing Values
                  ***************************

                        Row  Nmis   Columns
                          2     2   1 3
                          4     1   2
                          7     1   1
                         10     1   3

          Initial and Final Imputation of Missing Values
          **********************************************

                     Dense Matrix (12 by 3)

                    |          1          2          3
                  ----------------------------------------
                  1 |   42.800000  40.000000  37.000000
                  2 |   51.750000  93.500000  45.000000
                  3 |   37.500000  35.500000  34.000000
                  4 |   39.500000  37.000000  36.000000
                  5 |   45.500000  52.000000  43.000000
                  6 |   38.500000  17.000000  28.000000
                  7 |   41.150000  38.500000  37.000000
                  8 |   22.500000  8.5000000  20.000000
                  9 |   37.000000  33.000000  34.000000
                 10 |   23.500000  9.5000000  29.000000
                 11 |   33.000000  21.000000  38.000000
                 12 |   58.000000  79.000000  47.000000
```

```
dssq = ssq(data - xful);
print "KNEARN(k=2): ssq=",dssq;
```

```
KNEARN(k=2): ssq= 216.485
```

For such a small data set, the `"mindist"` option for starting values generates a slightly better result:

```
    optn = [ "print"            3 ,
             "ppatt"            2 ,
             "cent"               ,
             "scal"               ,
             "start"     "mindis" ,
             "pinit"            2 ,
             "seed"           123 ,
             "tol"          1.e-3 ,
             "maxit"           30 ];
    < xful,scal > = impute(b,"none",optn,.,bounds);


           Initial and Final Imputation of Missing Values
           **********************************************

                      Dense Matrix (12 by 3)

                    |           1          2          3
                   -------------------------------------
                  1 |   42.800000  40.000000  37.000000
                  2 |   58.000000  93.500000  47.000000
                  3 |   37.500000  35.500000  34.000000
                  4 |   39.500000  38.500000  36.000000
                  5 |   45.500000  52.000000  43.000000
                  6 |   38.500000  17.000000  28.000000
                  7 |   42.800000  38.500000  37.000000
                  8 |   22.500000  8.5000000  20.000000
                  9 |   37.000000  33.000000  34.000000
                 10 |   23.500000  9.5000000  20.000000
                 11 |   33.000000  21.000000  38.000000
                 12 |   58.000000  79.000000  47.000000


    dssq = ssq(data - xful);
    print "MINDIST: ssq=",dssq;


MINDIST: ssq= 211.540


   /* LINREG imputation and no Iteration:
      bounds does not enforce any constraints */
    optn = [ "print"            3 ,
             "ppatt"            2 ,
             "cent"               ,
             "scal"               ,
             "start"     "linreg" ,
```

```
          "knn"              2 ,
          "pinit"            2 ,
          "seed"           123 ,
          "tol"          1.e-3 ,
          "maxit"           30 ];
    < xful,scal > = impute(b,"none",optn,.,bounds);
```

The following output informs about the size of the data matrix used for estimating each column:

```
          Rows and Columns Used for Columnwise Estimation
          ***********************************************

Column    1 Cols : 2
Column    1 Rows :  1  3  5  6  8  9 10 11 12
Column    2 Cols : 1 3
Column    2 Rows :  1  3  5  6  8  9 11 12
Column    3 Cols : 2
Column    3 Rows :  1  3  5  6  7  8  9 11 12


           Initial and Final Imputation of Missing Values
           **********************************************

                    Dense Matrix (12 by 3)

              |          1          2          3
             -------------------------------------
             1 |   42.800000  40.000000  37.000000
             2 |   61.026443  93.500000  50.000000
             3 |   37.500000  35.500000  34.000000
             4 |   39.500000  41.540066  36.000000
             5 |   45.500000  52.000000  43.000000
             6 |   38.500000  17.000000  28.000000
             7 |   37.625282  38.500000  37.000000
             8 |   22.500000  8.5000000  20.000000
             9 |   37.000000  33.000000  34.000000
            10 |   23.500000  9.5000000  25.783453
            11 |   33.000000  21.000000  38.000000
            12 |   58.000000  79.000000  47.000000
```

```
    dssq = ssq(data - xful);
    print "LINREG: ssq=",dssq;
```

```
LINREG: ssq= 185.958
```

```
/* SIMPLS imputation and no Iteration:
   bounds does not enforce any constraints */
 optn = [ "print"            3 ,
          "ppatt"            2 ,
          "cent"               ,
          "scal"               ,
          "start"     "simpls" ,
          "nfact"            1 ,
          "knn"              2 ,
          "pinit"            2 ,
          "seed"           123 ,
          "tol"          1.e-3 ,
          "maxit"           30 ];
 < xful,scal > = impute(b,"none",optn,.,bounds);


        Initial and Final Imputation of Missing Values
        **********************************************

                    Dense Matrix (12 by 3)

                  |          1          2          3
                 -----------------------------------
                1 |   42.800000  40.000000  37.000000
                2 |   63.500000  93.500000  50.000000
                3 |   37.500000  35.500000  34.000000
                4 |   39.500000  37.125217  36.000000
                5 |   45.500000  52.000000  43.000000
                6 |   38.500000  17.000000  28.000000
                7 |   40.178718  38.500000  37.000000
                8 |   22.500000  8.5000000  20.000000
                9 |   37.000000  33.000000  34.000000
               10 |   23.500000  9.5000000  26.471869
               11 |   33.000000  21.000000  38.000000
               12 |   58.000000  79.000000  47.000000


 dssq = ssq(data - xful);
 print "SIMPLS: ssq=",dssq;
```

This is the best result we ever had. Even iteration from this start solution will not improve the result.

```
SIMPLS: ssq= 71.176
```

Iterative linear regression with bounds:

```
optn = [ "print"            3 ,
         "ppatt"            2 ,
         "cent"               ,
         "scal"               ,
         "start"     "mindis" ,
         "pinit"            2 ,
         "seed"           123 ,
         "tol"          1.e-3 ,
         "maxit"           30 ];
< xful,scal > = impute(b,"linreg",optn,.,bounds);
```

```
            Initial Imputation of Missing Values
            ************************************

                  Matrix: Dense Storage

              |           1          2          3
             ------------------------------------
             1 |   0.4865523  0.0417819  0.2145767
             2 |   1.9597784  2.0088737  1.5556812
             3 |  -0.0271384 -0.1236744 -0.1877546
             4 |   0.1667072 -0.0133702  0.0804663
             5 |   0.7482438  0.4829988  1.0192394
             6 |   0.0697844 -0.8038838 -0.9924173
             7 |   0.4865523 -0.0133702  0.2145767
             8 |  -1.4809799 -1.1164124 -2.0653009
             9 |  -0.0755998 -0.2155946 -0.1877546
            10 |  -1.3840571 -1.0796443 -2.0653009
            11 |  -0.4632908 -0.6568115  0.3486872
            12 |   1.9597784  1.4757367  1.5556812


              ********************************
              Iteration History for Imputation
              ********************************

           Iter   MaxChange   L1_Change   L2_Change
              1  0.72423033  1.29187591  0.72953932
              2  0.11266522  0.17170317  0.01525927
              3  0.05306570  0.06668709  0.00299848
              4  0.02415771  0.02891577  6.062e-004
              5  0.01086942  0.01281619  1.219e-004
              6  0.00487155  0.00571724  2.444e-005
              7  0.00218048  0.00255512  4.892e-006
```

```
 8  9.755e-004  0.00114255  9.790e-007

Final Imputation of Missing Values
**********************************

      Dense Matrix (12 by 3)

  |           1          2          3
--------------------------------------
1 |   42.800000  40.000000  37.000000
2 |   59.864765  93.500000  50.000000
3 |   37.500000  35.500000  34.000000
4 |   39.500000  41.940535  36.000000
5 |   45.500000  52.000000  43.000000
6 |   38.500000  17.000000  28.000000
7 |   38.247677  38.500000  37.000000
8 |   22.500000  8.5000000  20.000000
9 |   37.000000  33.000000  34.000000
10 |  23.500000  9.5000000  25.932500
11 |  33.000000  21.000000  38.000000
12 |  58.000000  79.000000  47.000000
```

```
dssq = ssq(data - xful);
print "LINREG: ssq=",dssq;
```

```
LINREG: ssq= 194.920
```

The bounded linear regression improved the result slightly. But even the unbounded imputation would still result in `ssq= 201.699` after 16 iterations.
The best result can be obtained using PLS with only one or two factors:

```
optn = [ "print"           3 ,
         "start"    "randnrm" ,
         "seed"         123 ,
         "tol"        1.e-3 ,
         "maxit"         30 ,
         "nfact"          2 ];
< xful,scal > = impute(b,"simpls",optn);
```

```
  Number Observations . . . . . .          12
  Number Variables. . . . . .           3
```

Number Missing Values	5
Percentage of Missing Values	13.89
Seed for Random Generator	123
Maximum Number Iterations	30
Termination Tolerance	0.00100000
Number PLS Factors	2

Column Summary for Imputation Method

Col	Variable	Nmiss	Mean	StdDev	LowRange	UppRange
1	v1	2	37.780000	10.317493	22.500000	58.000000
2	v2	1	38.863636	27.197510	8.5000000	93.500000
3	v3	2	35.400000	7.4565408	20.000000	47.000000

Initial and Final Imputation Methods

Column	Variable	Initial	Final
1	v1	RANDNRM	SIMPLS
2	v2	RANDNRM	SIMPLS
3	v3	RANDNRM	SIMPLS

Pattern of 5 Missing Values

Row	Nmis	Columns
2	2	1 3
4	1	2
7	1	1
10	1	3

Iteration History for Imputation

Iter	MaxChange	L1_Change	L2_Change
1	21.5395661	40.9055125	652.946376
2	6.40766223	11.6881662	64.8537284
3	5.28965718	9.23530583	36.8287220

```
 4  3.45803925  6.03102579  15.4982981
 5  2.04427893  3.65635643  5.55493462
 6  1.17143809  2.15554386  1.88838495
 7  0.66956523  1.26213316  0.63799558
 8  0.38512072  0.74004642  0.21743608
 9  0.22323186  0.43559441  0.07494784
10  0.13029623  0.25742080  0.02609413
11  0.07648746  0.15264598  0.00915750
12  0.04510553  0.09076018  0.00323319
13  0.02669521  0.05407533  0.00114670
14  0.01584410  0.03226861  4.081e-004
15  0.00942476  0.01927855  1.456e-004
16  0.00561611  0.01152813  5.204e-005
17  0.00335121  0.00689831  1.863e-005
18  0.00200190  0.00413006  6.676e-006
19  0.00119691  0.00247371  2.395e-006
20  7.161e-004  0.00148211  8.596e-007

Final Imputation of Missing Values
**********************************

      Dense Matrix (12 by 3)

  |           1          2          3
-------------------------------------
1 |   42.800000  40.000000  37.000000
2 |   65.337167  93.500000  54.767629
3 |   37.500000  35.500000  34.000000
4 |   39.500000  37.465833  36.000000
5 |   45.500000  52.000000  43.000000
6 |   38.500000  17.000000  28.000000
7 |   40.388498  38.500000  37.000000
8 |   22.500000  8.5000000  20.000000
9 |   37.000000  33.000000  34.000000
10 |   23.500000  9.5000000  25.865268
11 |   33.000000  21.000000  38.000000
12 |   58.000000  79.000000  47.000000

    dssq = ssq(data - xful);
    print "SIMPLS: ssq=",dssq;

SIMPLS: ssq= 105.760

    optn = [ "print"            3 ,
```

```
        "start"    "randnrm" ,
        "seed"          123 ,
        "tol"         1.e-3 ,
        "maxit"          30 ,
        "nfact"           1 ];
< xful,scal > = impute(b,"simpls",optn);
```

****
Iteration History for Imputation
****

Iter	MaxChange	L1_Change	L2_Change
1	21.8275912	42.6705300	693.665397
2	5.98241768	12.5443747	63.7278783
3	3.11618425	7.29110268	19.5805112
4	1.71686479	4.06547379	6.13956155
5	0.97019309	2.29636040	1.98675193
6	0.55581194	1.31354141	0.65733591
7	0.32348564	0.75799156	0.22055585
8	0.18921333	0.43988076	0.07464165
9	0.11078408	0.25616883	0.02539258
10	0.06489844	0.14950288	0.00866570
11	0.03802805	0.08736618	0.00296302
12	0.02228534	0.05109621	0.00101433
13	0.01306000	0.02989878	3.475e-004
14	0.00765346	0.01750078	1.191e-004
15	0.00448491	0.01024594	4.083e-005
16	0.00262803	0.00599939	1.400e-005
17	0.00153987	0.00351320	4.803e-006
18	9.022e-004	0.00205745	1.647e-006

Final Imputation of Missing Values
****

Dense Matrix (12 by 3)

	1	2	3
1	42.800000	40.000000	37.000000
2	65.364404	93.500000	54.771720
3	37.500000	35.500000	34.000000
4	39.500000	37.473434	36.000000
5	45.500000	52.000000	43.000000
6	38.500000	17.000000	28.000000
7	40.297499	38.500000	37.000000

```
 8 |   22.500000  8.5000000  20.000000
 9 |   37.000000  33.000000  34.000000
10 |   23.500000  9.5000000  25.866265
11 |   33.000000  21.000000  38.000000
12 |   58.000000  79.000000  47.000000

SIMPLS: ssq= 106.489
```

3.5 Function `nnmf`

<u,h> = nnmf(a,k<optn<,uini<,hini>>>)

<u,h,d> = nnmf(a,k<optn<,uini<,hini>>>)

Purpose: The `nnmf` function implements the following three algorithms (including modifications) for non-negative matrix factorization:

- the common alternating least squares (ALS) algorithm (see e.g. Hoyer, 2004);
- the *gdcls* (*gradient descent with constrained least squares*) algorithm (Shanaz, Berry, Pauca, & Plemmons, 2004)
- the Lee & Seung (1999, 2001) algorithm.

There are two versions of the factorization model: For a given (often sparse) $n \times m$ matrix $\mathbf{A}$ which should have only nonnegative entries we estimate

1. the $n \times k$ matrix $\mathbf{U}$ and the $m \times k$ matrix $\mathbf{H}$

   $$\mathbf{A} = \mathbf{U}\mathbf{H}^T$$

   with the k columns of $\mathbf{H}$ normalized to unit length,
2. the $n \times k$ matrix $\mathbf{U}$, the $m \times k$ matrix $\mathbf{H}$, and the $k \times k$ matrix $\mathbf{D}$

   $$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{H}^T$$

   where the k columns of both matrices $\mathbf{U}$ and $\mathbf{H}$ are normalized to unit length,

and subject to the constraint that the two matrices $\mathbf{U}$ and $\mathbf{H}$ have only nonnegative entries. The restriction implies that the resulting matrices $\mathbf{U}$ and $\mathbf{H}$ usually are sparse. The model is invariant w.r.t. the column order: In our implementation the columns are ordered:

1. for the $\mathbf{A} = \mathbf{U}\mathbf{H}^T$ model with respect to descending $L2$ norms of the columns of $\mathbf{U}$

2. for the $\mathbf{A} = \mathbf{UDH}^T$ model with respect to decreasing diagonal values of $\mathbf{D}$

Unfortunately, this gradient descent algorithm produces solutions **U** and **H** which are local and not necessarily global optima. Different starting values for the **U** and **H** matrices will usually result not only in different values for the L_2 and L_1 errors

$$f^{L2} = ||(A - UH^T)||_2^2 \quad f^{L1} = \max(A - UH^T)$$

it will also result in different parameter matrices **U** and **H** even in applications where the L_2 and L_1 errors maybe very close. See the results for the Reuters data set below.

If the data set has missing values, an EM like modified algorithm is used for the iterative imputation of missing values satisfying the model expectation. This modified algorithm may have some convergence problems especially with small data sets and/or many missing values.

Convergence problems may have to do with different scaled data. That means, we found examples where the algorithm did not converge for unscaled data, but was converging for standardized data. This is especially so when the data have too many missing values.

Input: **a** is a given $n \times k$ (usually sparse) matrix.

k is the number of factors, i.e. the column dimension of matrices **U** and **H**.

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.

uini is an optional starting solution for **U**

hini is an optional starting solution for **H**

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

Option	Second Column	Meaning
"bet_h"	real	only for ACLS and MACLS: H parameter β
"bet_w"	real	only for ACLS and MACLS: W parameter β
"cstrt"		compute starting values using the centroid method
"lam_h"	real	only for ACLS and MACLS: H parameter λ
"lam_w"	real	only for ACLS and MACLS: W parameter λ
"lambda"	real	only for GCLS and MGCLS: regularization parameter λ (default=10^{-2})
"meth"	string	algorithm used for estimation
	"acls"	alternative constrained least squares (LS)
	"gcls"	gradient descent constrained least squares
	"macls"	modified alternative constrained LS
	"mgcls"	modified gradient descent constrained LS
	"le_se"	Lee and Seung (2001) algorithm
	"ga_w"	genetic algorithm using modified ACLS
"maxit"	int	maximum number of iterations (def: 400)
"print"	int	amount of printed output (=0: no output)
"redu"		modification of ACLS and GCLS algorithm: if truncation w.r.t. the sparsity cutoff "spar" results in iterates that do not reduce the $L2$ fit criterion, the cutoff value is temporarily reduced to zero.
"scal"		scale all variables w.r.t. standard deviation
"seed"	int	seed for random number generator (default is day time)
"spar"	real	only for ACLS and GCLS: must be ≥ 0 (default=10^{-4})
		cutoff value for chopping off small and negative entries
"sstrt"		compute starting values using the common SVD
"tol"	real	termination threshold for the difference of the L_2 error values in succeeding iterations (default=10^{-4})

Shanaz, Berry, Pauca, & Plemmons (2004) recommend three separate solutions for different values of the regularization parameter $\lambda = .1, .01, .001$

The methods "acls" and "gcls" use unrestricted least squares estimation of $\mathbf{U}$ and $\mathbf{H}$ in each iteration. Afterward negative and small positive, i.e. $\leq$ optn[7] entries of the unrestricted LS estimates are set to zero. Especially in later iterations that may not reduce the LS fit criterion and can lead to some unsatisfying convergence behavior. If after some iteration this approach increases the L2 fit criterion at an iteration, then for optn[3]=1 the algorithm tries temporarily to reduce the sparsity threshold up to zero.

The methods "macls" and "mgcls" use a different approach. Starting at iterations where set-to-zero unrestricted LS estimates do not improve the

fit in one iteration, an interior point algorithm is used for the nonnegative constrained linear least squares solution. This is indicated in the iteration history with an asterisk. The methods `"acls"` and `"gcls"` must not necessarily converge, but the methods `"macls"`, `"mgcls"`, and `"le_se"` should always converge, at least if there are no missing values in the data set. When the data set has missing values, the `"macls"` or `"mgcls"` methods should be used usually with the `"sstrt"` or `"cstr"` option specifying better than rand starting values for the parameters.

Output: **u** is a $n \times k$ (sometimes sparse) matrix **U** with nonnegative entries.

h is a $m \times k$ (sometimes sparse) matrix **H** with nonnegative entries and columns scaled to unit norm.

d is a $k \times k$ diagonal matrix with nonnegative entries. In that case, both matrices, **U** and **H** have unit column norms.

Restrictions: 1. The model for non-negative matrix factorization assumes implicitly that the input data has only nonnegative real or integer values. No negative values, complex or string data are permitted in `a`. That means, data cannot be centerd.

2. No missing values for `uini` and `hini` are permitted.
3. The number of factors `k` specified in the second input argument must be smaller than the row and column dimension of the input matrix `a`, $k < \min(b, m)$.
4. The convergence for a gradient descent algorithm becomes slow for small values of the termination criterion.

Relationships: svd(), eig(), pca()

Examples: 1. Small Example with 29×9 matrix:

```
v =  [ 4  1  1,   7  1  1,   3  1  1,   5  1  1,   1  1  1,
       6  1  1,   2  1  2,   7  2  1,   9  2  1,  10  2  1,
      11  2  1,   8  2  1,   2  2  2,  13  3  1,  12  3  1,
      14  3  1,  11  3  1,   9  4  1,  15  4  1,   8  4  2,
       2  4  1,  16  4  1,   7  5  1,   9  5  1,  17  5  1,
      18  5  1,  10  5  1,  19  5  1,   2  5  2,  21  6  1,
       7  6  1,  12  6  1,  23  6  1,   3  6  1,  14  6  1,
      24  6  1,  22  6  1,   2  6  1,  20  6  1,   7  7  1,
       9  7  1,  17  7  1,  18  7  1,  15  7  1,  19  7  1,
       8  7  1,   2  7  1,   5  8  1,  15  8  1,  25  8  1,
      26  8  1,  28  8  1,   2  8  2,  27  8  1,  29  9  1,
      25  9  1,  26  9  1,   2  9  1 ];


nr = 29; nc = 9; nf = 2;
a = spmat(nr,nc,v[,1],v[,2],v[,3]);
```

```
rnam = [ "Row1" : "Row29" ];
cnam = [ "Col1" : "Col9" ];
a = rname(a,rnam);
a = cname(a,cnam);
print a;
attrib(a);
```

```
---------------------------------------
         Table of Attributes
---------------------------------------
Object Name      name               a
Object Type      otyp       matrix_gen
Data Type        dtyp             real
Storage Type     styp        spar_full
Row Names        rnam               29
Column Names     cnam                9
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow               29
Number Columns   ncol                9
Lower Bandwidth  lbw                20
Upper Bandwidth  ubw                 7
Size in Bytes    size             1300
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer               58
Smallest Value   vmin                0
Largest Value    vmax                2
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
---------------------------------------
```

```
optn = [ "print"          2 ,      /* ipri */
         "meth"      "acls" ,      /* imet=ACLS */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< w,h > = nnmf(a,nf,optn);
```

```
print "Matrix W:", w;
print "Matrix H:", h;
```

```
                Nonnegative Matrix Factorization
                Alternative Least Squares Method

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  31.1622928  41.8377072  59.3271077  1.56707573
          2  29.5668208  1.59547195  58.2275943  1.41021080
          3  29.5304610  0.03635983  58.3888360  1.37767297
          4  29.5202314  0.01022956  58.5079174  1.37839380
          5  29.5158945  0.00433692  58.5720213  1.38318641
          6  29.5128751  0.00301941  58.5865733  1.38781491
          7  29.5107817  0.00209344  58.6070979  1.39143978
          8  29.5095839  0.00119775  58.6160205  1.39400452
          9  29.5088621  7.218e-004  58.6206834  1.39581107
         10  29.5084071  4.550e-004  58.6229120  1.39708706
         11  29.5081119  2.952e-004  58.6241040  1.39798875
         12  29.5079163  1.956e-004  58.6248068  1.39862564
         13  29.5077847  1.316e-004  58.6252513  1.39907521
         14  29.5076951  8.961e-005  58.6255454  1.39939236

       L2 Precision : 29.5077   L1 Precision : 58.6255
                  Computer Time : 0 seconds
```

```
Matrix W:
   |         1         2
-------------------------
 1 |   0.24548   0.29669
 2 |   4.00176   1.19839
 3 |   0.14287   1.20987
 4 |   0.24548   0.29669
 5 |   0.62826   0.25368
 6 |   0.24548   0.29669
 7 |   1.51058   1.10511
 8 |   1.62851   0.00000
 9 |   1.74124   0.00000
10 |   0.93368   0.00000
11 |   0.41529   0.28223
12 |   0.00000   1.19911
13 |   0.00000   0.28594
14 |   0.00000   1.19911
15 |   1.19034   0.00000
16 |   0.37353   0.00000
17 |   0.92029   0.00000
18 |   0.92029   0.00000
19 |   0.92029   0.00000
20 |   0.00000   0.91317
21 |   0.00000   0.91317
22 |   0.00000   0.91317
23 |   0.00000   0.91317
24 |   0.00000   0.91317
25 |   0.55572   0.00000
26 |   0.55572   0.00000
27 |   0.38278   0.00000
28 |   0.38278   0.00000
29 |   0.17294   0.00000
```

```
Matrix H:
   |         1         2
-------------------------
 1 |   0.27882   0.32428
 2 |   0.44701   0.04656
 3 |   0.00000   0.28233
 4 |   0.36881   0.00000
 5 |   0.48038   0.00238
 6 |   0.00000   0.90164
 7 |   0.42855   0.00000
 8 |   0.37795   0.00000
 9 |   0.17076   0.00000
```

Scaling $\mathbf{W}$ to unit column norm gives $\mathbf{D}$:

```
Matrix D:
 D |         1         2
-------------------------
 1 |   5.53695         0
 2 |         0   3.41412
```

```
optn = [ "print"          2 ,      /* ipri */
         "meth"      "acls" ,      /* imet=ACLS */
         "reduc"            ,      /* ivers: red stepwise */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< w,h > = nnmf(a,nf,optn);
print "Matrix W:", w;
print "Matrix H:", h;
```

```
                Alternative Least Squares Method

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  31.1622928  41.8377072  59.3271077  1.56707573
          2  29.5668208  1.59547195  58.2275943  1.41021080
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          3  29.5304610  0.03635983  58.3888360  1.37767297
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          4  29.5202314  0.01022956  58.5079174  1.37839380
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          5  29.5158945  0.00433692  58.5720213  1.38318641
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          6  29.5128751  0.00301941  58.5865733  1.38781491
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          7  29.5107817  0.00209344  58.6070979  1.39143978
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          8  29.5095839  0.00119775  58.6160205  1.39400452
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          9  29.5088621  7.218e-004  58.6206834  1.39581107
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
         10  29.5084071  4.550e-004  58.6229120  1.39708706
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
         11  29.5081119  2.952e-004  58.6241040  1.39798875
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
         12  29.5079163  1.956e-004  58.6248068  1.39862564
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
         13  29.5077847  1.316e-004  58.6252513  1.39907521
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
         14  29.5076951  8.961e-005  58.6255454  1.39939236

        L2 Precision : 29.5077   L1 Precision : 58.6255
                  Computer Time : 0 seconds


optn = [ "print"          2 ,      /* ipri */
         "meth"      "gcls" ,      /* imet=GCLS */
         "lambda"     1.e-2 ,      /* lambda */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< w,h > = nnmf(a,nf,optn);
print "Matrix W:", w;
print "Matrix H:", h;
```

```
                Nonnegative Matrix Factorization
              Gradient Descent with Constrained LS

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  31.8199616  41.1800384  61.1333298  1.58633932
          2  29.5795656  2.24039607  57.8102814  1.44247273
          3  29.5185176  0.06104796  58.1913897  1.40570010
          4  29.5048637  0.01365387  58.3210556  1.39913255
          5  29.5006117  0.00425202  58.3984935  1.39981437
          6  29.4996473  9.644e-004  58.4560351  1.40187165
          7  29.4993103  3.370e-004  58.5081693  1.40380996
          8  29.4991104  1.999e-004  58.5342738  1.40530487
          9  29.4989337  1.767e-004  58.5469363  1.40645136
         10  29.4987784  1.553e-004  58.5534470  1.40733370
         11  29.4986517  1.266e-004  58.5570644  1.40801370
         12  29.4985536  9.815e-005  58.5592467  1.40853859

        L2 Precision : 29.4986   L1 Precision : 58.5592
                  Computer Time : 0 seconds


optn = [ "print"          2 ,      /* ipri */
         "meth"      "gcls" ,      /* imet=GCLS */
         "reduc"            ,      /* ivers: red stepwise */
         "lambda"     1.e-2 ,      /* lambda */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< w,h > = nnmf(a,nf,optn);
print "Matrix W:", w;
print "Matrix H:", h;

                Nonnegative Matrix Factorization
              Gradient Descent with Constrained LS

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  31.8199616  41.1800384  61.1333298  1.58633932
          2  29.5795656  2.24039607  57.8102814  1.44247273
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          3  29.5185176  0.06104796  58.1913897  1.40570010
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          4  29.5048637  0.01365387  58.3210556  1.39913255
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
```

```
 5  29.5006117  0.00425202  58.3984935  1.39981437
 [H]: Reduce Sparsity Bound: from 0.0001 to 0
 6  29.4996473  9.644e-004  58.4560351  1.40187165
 [H]: Reduce Sparsity Bound: from 0.0001 to 0
 7  29.4993103  3.370e-004  58.5081693  1.40380996
 [H]: Reduce Sparsity Bound: from 0.0001 to 0
 8  29.4991104  1.999e-004  58.5342738  1.40530487
 [H]: Reduce Sparsity Bound: from 0.0001 to 0
 9  29.4989337  1.767e-004  58.5469363  1.40645136
 [H]: Reduce Sparsity Bound: from 0.0001 to 0
10  29.4987784  1.553e-004  58.5534470  1.40733370
 [H]: Reduce Sparsity Bound: from 0.0001 to 0
11  29.4986517  1.266e-004  58.5570644  1.40801370
 [H]: Reduce Sparsity Bound: from 0.0001 to 0
12  29.4985536  9.815e-005  58.5592467  1.40853859

 L2 Precision : 29.4986   L1 Precision : 58.5592
           Computer Time : 0 seconds
```

The modified ACLS method:

```
optn = [ "print"          2 ,      /* ipri */
         "meth"     "macls" ,      /* imet=MACLS */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< w,h > = nnmf(a,nf,optn);
print "Matrix W:", w;
print "Matrix H:", h;
```

```
            Nonnegative Matrix Factorization
         Modified Alternative Least Squares Method

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  31.1622928  41.8377072  59.3271077  1.56707573
          2  29.5668208  1.59547195  58.2275943  1.41021080
    *     3  29.4839889  0.08283192  58.0021021  1.44327008
    *     4  29.4639349  0.02005401  57.9007044  1.45688898
    *     5  29.4598865  0.00404842  57.9160526  1.46371542
    *     6  29.4585996  0.00128688  57.9489543  1.46707511
    *     7  29.4579649  6.347e-004  57.9615878  1.46892086
    *     8  29.4576218  3.431e-004  57.9675583  1.47011678
```

```
    *     9  29.4574348  1.871e-004  57.9709219  1.47095374
    *    10  29.4573331  1.017e-004  57.9730487  1.47155810
    *    11  29.4572781  5.505e-005  57.9744815  1.47199947

        L2 Precision : 29.4573   L1 Precision : 57.9745
                  Computer Time : 0 seconds
```

```
optn = [ "print"          2 ,      /* ipri */
         "meth"     "mgcls" ,      /* imet=MGCLS */
         "lambda"     1.e-2 ,      /* lambda */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< w,h > = nnmf(a,nf,optn);
print "Matrix W:", w;
print "Matrix H:", h;
```

```
              Nonnegative Matrix Factorization
         Modified Gradient Descent with Constrained LS

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  31.8199616  41.1800384  61.1333298  1.58633932
          2  29.5795656  2.24039607  57.8102814  1.44247273
    *     3  29.4852749  0.09429062  57.7222685  1.45876048
    *     4  29.4605989  0.02467602  57.7961402  1.46515426
    *     5  29.4581430  0.00245590  57.9059178  1.46838170
    *     6  29.4576509  4.921e-004  57.9503213  1.46996203
    *     7  29.4574447  2.063e-004  57.9654872  1.47089258
    *     8  29.4573380  1.067e-004  57.9712919  1.47152400
    *     9  29.4572806  5.731e-005  57.9738880  1.47197621

        L2 Precision : 29.4573   L1 Precision : 57.9739
                  Computer Time : 0 seconds
```

Random start supplied: the function will use the `abs` function for converting negative starting values into positive ones:

```
uini = rand(nr,nf);
hini = rand(nf,nc);


wini = rand(nr,nf);
hini = rand(nc,nf);
```

```
optn = [ "print"          2 ,      /* ipri */
         "meth"      "acls" ,      /* imet=ACLS */
         "reduc"            ,      /* ivers: red stepwise */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< w,h > = nnmf(a,nf,optn,wini,hini);
print "Matrix W:", w;
print "Matrix H:", h;
```

This is obviously not the same solution than that above. It is obvious that the algorithm is sensitive toward local minima which are not global minima:

```
                Nonnegative Matrix Factorization
                Alternative Least Squares Method

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  38.0831139  34.9168861  72.5468889  1.31749406
          2  30.3735292  7.70958470  60.3504187  1.38422222
          3  29.6422401  0.73128916  58.6481741  1.42850845
          4  29.5260888  0.11615128  58.5071396  1.43382507
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          5  29.5084073  0.01768152  58.5656385  1.42483825
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          6  29.5069248  0.00148249  58.6169301  1.41664569
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          7  29.5064085  5.163e-004  58.6283865  1.41158754
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
          8  29.5063642  4.430e-005  58.6295958  1.40818692

       L2 Precision : 29.5064   L1 Precision : 58.6296
                 Computer Time : 0 seconds
```

The Lee & Seung (2001) algorithm shows very slow convergence even for a relaxed termination tolerance:

```
optn = [ "print"          2 ,      /* ipri */
         "meth"     "le_se" ,      /* imet=Lee&Seung */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-2 ,      /* term */
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
```

```
< w,h > = nnmf(a,nf,optn);
print "Matrix W:", w;
print "Matrix H:", h;
```

Nonnegative Matrix Factorization
Lee and Seung Gradient Descent Method

Iter	OptCrit	DiffCrit	L1Crit	LooCrit
1	38.3592786	60.4301413	74.1290673	1.46756005
2	33.7601177	4.59916096	69.0975009	1.46833393
3	31.6953061	2.06481162	65.9720404	1.41134940
4	30.7264248	0.96888127	63.3798045	1.36190252
5	30.2942686	0.43215619	61.9009471	1.33597122
6	30.0650234	0.22924515	61.1395125	1.32256924
7	29.9227252	0.14229828	60.7367344	1.31447365
8	29.8252541	0.09747104	60.5172517	1.30879957
9	29.7563857	0.06886839	60.3788893	1.30443531
10	29.7081454	0.04824037	60.2584841	1.30084890
11	29.6736810	0.03446440	60.1376416	1.29767841
12	29.6478035	0.02587750	60.0187795	1.29467530
13	29.6272128	0.02059070	59.9082436	1.29169594
14	29.6097736	0.01743916	59.8075402	1.28865580
15	29.5940291	0.01574450	59.7155045	1.28550330
16	29.5789513	0.01507781	59.6300223	1.28220860
17	29.5637915	0.01515977	59.5487319	1.27875694
18	29.5479900	0.01580147	59.4692500	1.27514453
19	29.5311228	0.01686729	59.3892218	1.27137627
20	29.5128735	0.01824926	59.3063290	1.26746457
21	29.4930237	0.01984983	59.2183032	1.26342877
22	29.4714519	0.02157176	59.1229692	1.25929452
23	29.4481424	0.02330946	59.0183541	1.25509307
24	29.4231931	0.02494937	58.9028339	1.25085995
25	29.3968178	0.02637529	58.7753120	1.24663315
26	29.3693514	0.02746640	58.6354611	1.24245013
27	29.3412191	0.02813232	58.4848978	1.23834752
28	29.3129170	0.02830207	58.3339429	1.23435456
29	29.2849686	0.02794840	58.1747134	1.23049340
30	29.2578790	0.02708960	58.0103011	1.22677725
31	29.2320931	0.02578588	57.8442990	1.22321021
32	29.2079649	0.02412818	57.6804697	1.21978837
33	29.1857423	0.02222263	57.5224043	1.21650189
34	29.1655672	0.02017508	57.3732240	1.21333770
35	29.1474877	0.01807954	57.2353598	1.21028212

```
 36  29.1314756  0.01601207  57.1104313  1.20732319
 37  29.1174458  0.01402984  56.9992240  1.20445222
 38  29.1052722  0.01217359  56.9017484  1.20166455
 39  29.0948004  0.01047177  56.8173720  1.19895964
 40  29.0858632  0.00893722  56.7449909  1.19634041

L2 Precision : 29.0859   L1 Precision : 56.745
          Computer Time : 0 seconds
```

2. (Weighted) Reuters data set with 5871×9 matrix:

```
options NOECHO;
%inc "../tdata/reuters.dat";
options ECHO;
datb = shape(data,.,3);
print "Maxima=", max = datb[ <>, ];

nr = 5871; nc = max[2]; nf = 2;
datc = spmat(nr,nc,datb[,1],datb[,2],datb[,3]);
attrib(datc);
```

```
Maxima=
   |         1         2         3
-----------------------------------
 1 |   5871.00    175.00      3.16
```

```
--------------------------------------
         Table of Attributes
--------------------------------------
Object Name      name             datc
Object Type      otyp       matrix_gen
Data Type        dtyp             real
Storage Type     styp        spar_full
Row Names        rnam                0
Column Names     cnam                0
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow             5871
Number Columns   ncol              175
Lower Bandwidth  lbw              5774
Upper Bandwidth  ubw               163
Size in Bytes    size           190652
String Length    slen                0
```

```
Number Strings   nstr               0
Number MissVals  nmis               0
Number NonzeroV  nzer           10432
Smallest Value   vmin               0
Largest Value    vmax           3.155
Frobenius Norm   nrm2               .
Recip Condition  rcond              .
Determinant      det                .
Largest SingVal  svb                .
Smallest SingV   svs                .
Num Rank Estim   rnk                .
---------------------------------------
```

/*— Compute SVD with svdtrip() ———————*/ /* svd() for complete SVD is too time consuming for (nr,nc)! */

We can compute the two largest singular values of the 5871×175 matrix using the `svdtrip()` function with Block Lanczos method:

```
optn = cons(12,1,.);
optn[ 1] =  2;      /* ipri */
optn[ 2] =  2;      /* ntri: number values */
optn[ 4] =  1;      /* iata: no A'A */
optn[ 5] = 50;      /* nsub: maximum subspace dim */
optn[ 6] = 10;      /* nblk: initial block size */
optn[ 8] = 1.e-6;   /* ctol: tolerance */
 < D,V,U > = svdtrip(datc,"bls",optn);
 print "SingValues=", D;
```

```
                Singular Values and Residual Norms
                **********************************

                    1 1.72064576e+001   1.4125e-013
                    2 1.14811068e+001   1.3349e-013
```

The Lee & Seeung (2001) algorithm converges slowly but to a rather good solution:

```
optn = [ "print"          2 ,      /* ipri */
         "meth"     "le_se" ,      /* imet=Lee&Seung */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-6 ,      /* spar: this must be small */
         "seed"          5 ];      /* seed */
< w,h,d > = nnmf(datc,nf,optn);
```

```
print "Matrix D:", d;
attrib(w);
attrib(h);
```

Nonnegative Matrix Factorization
Lee and Seung Gradient Descent Method

Iter	OptCrit	DiffCrit	L1Crit	LooCrit
1	2439.67020	313480.973	8316.84172	2.96536805
2	2420.43798	19.2322189	8273.38962	2.91453962
3	2403.44247	16.9955081	8261.51290	2.84520774
4	2388.08459	15.3578787	8214.36625	2.77103399
5	2376.33346	11.7511313	8152.49231	2.72005360
6	2367.58884	8.74462784	8107.78849	2.70647537
7	2360.54641	7.04242724	8083.56720	2.72335250
8	2354.18695	6.35945428	8075.20952	2.75728293
9	2348.22053	5.96642442	8075.99090	2.79755713
10	2342.87260	5.34792582	8077.62445	2.83738912
11	2338.19843	4.67416908	8076.88156	2.87298242
12	2334.13554	4.06289611	8073.08368	2.90294414
13	2330.73285	3.40268616	8064.66794	2.92731496
14	2328.06581	2.66704492	8050.71233	2.94657622
15	2326.10827	1.95753981	8034.67445	2.96122892
16	2324.75693	1.35134077	8020.40237	2.97190029
17	2323.87213	0.88480091	8009.34414	2.97937623
18	2323.31091	0.56121433	8001.21947	2.98445198
19	2322.95227	0.35864614	7995.51473	2.98781865
20	2322.71687	0.23540066	7991.58544	2.99002701
21	2322.55900	0.15786100	7988.78545	2.99147962
22	2322.44969	0.10931378	7986.71974	2.99244439
23	2322.37088	0.07880865	7985.20053	2.99309122
24	2322.31210	0.05878365	7984.07563	2.99352904
25	2322.26707	0.04502515	7983.22180	2.99382904
26	2322.23185	0.03522457	7982.54621	2.99403724
27	2322.20386	0.02798912	7981.99206	2.99418331
28	2322.18135	0.02251088	7981.52277	2.99428670
29	2322.16321	0.01814198	7981.11363	2.99436029
30	2322.14855	0.01465366	7980.74174	2.99441271
31	2322.13666	0.01189129	7980.39292	2.99444986
32	2322.12693	0.00973323	7980.06352	2.99447592
33	2322.11888	0.00804432	7979.75597	2.99449382
34	2322.11215	0.00673782	7979.47304	2.99450559
35	2322.10644	0.00570190	7979.21479	2.99451277

36	2322.10157	0.00486959	7978.98210	2.99451645
37	2322.09736	0.00421399	7978.77214	2.99451752
38	2322.09370	0.00365607	7978.58335	2.99451671
39	2322.09051	0.00319325	7978.41216	2.99451461
40	2322.08771	0.00280353	7978.25773	2.99451171
41	2322.08523	0.00248238	7978.11698	2.99450840
42	2322.08300	0.00222069	7977.98786	2.99450500
43	2322.08101	0.00199077	7977.87114	2.99450174
44	2322.07921	0.00180513	7977.76432	2.99449876
45	2322.07756	0.00164494	7977.66725	2.99449617
46	2322.07604	0.00152019	7977.57820	2.99449404
47	2322.07465	0.00139519	7977.49745	2.99449244
48	2322.07337	0.00127806	7977.42323	2.99449138
49	2322.07219	0.00117786	7977.35431	2.99449090
50	2322.07111	0.00108206	7977.28968	2.99449103
51	2322.07012	9.894e-004	7977.22880	2.99449180
52	2322.06922	9.000e-004	7977.17025	2.99449326
53	2322.06840	8.167e-004	7977.11283	2.99449543
54	2322.06767	7.314e-004	7977.05760	2.99449832
55	2322.06702	6.540e-004	7977.00310	2.99450189
56	2322.06644	5.832e-004	7976.94930	2.99450614
57	2322.06592	5.152e-004	7976.89676	2.99451101
58	2322.06547	4.553e-004	7976.84514	2.99451646
59	2322.06506	4.023e-004	7976.79466	2.99452244
60	2322.06471	3.562e-004	7976.74545	2.99452888
61	2322.06439	3.181e-004	7976.69755	2.99453574
62	2322.06411	2.835e-004	7976.65152	2.99454298
63	2322.06385	2.560e-004	7976.60719	2.99455056
64	2322.06362	2.334e-004	7976.56475	2.99455846
65	2322.06340	2.150e-004	7976.52418	2.99456668
66	2322.06320	2.007e-004	7976.48535	2.99457519
67	2322.06301	1.873e-004	7976.44877	2.99458400
68	2322.06284	1.770e-004	7976.41405	2.99459311
69	2322.06267	1.691e-004	7976.38082	2.99460252
70	2322.06251	1.600e-004	7976.35006	2.99461225
71	2322.06235	1.526e-004	7976.32087	2.99462229
72	2322.06221	1.454e-004	7976.29330	2.99463265
73	2322.06207	1.382e-004	7976.26725	2.99464336
74	2322.06194	1.309e-004	7976.24263	2.99465440
75	2322.06182	1.235e-004	7976.21931	2.99466580
76	2322.06170	1.160e-004	7976.19716	2.99467755
77	2322.06159	1.086e-004	7976.17603	2.99468967
78	2322.06149	1.014e-004	7976.15581	2.99470216

```
 79  2322.06139  9.704e-005  7976.13620  2.99471502

L2 Precision : 2322.06   L1 Precision : 7976.14
          Computer Time : 9 seconds
```

The large **W** matrix has only 2331 nonzeros:

```
--------------------------------------
         Table of Attributes
--------------------------------------
Object Name      name               w
Object Type      otyp       matrix_low
Data Type        dtyp             real
Storage Type     styp      spar_packed
Row Names        rnam                0
Column Names     cnam                0
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow             5871
Number Columns   ncol                2
Lower Bandwidth  lbw              5870
Upper Bandwidth  ubw                 0
Size in Bytes    size            61036
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer             2331
Smallest Value   vmin                0
Largest Value    vmax            2.386
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
--------------------------------------
```

The smaller **H** matrix has 302 nonzeros:

```
--------------------------------------
         Table of Attributes
--------------------------------------
Object Name      name               h
Object Type      otyp       matrix_gen
Data Type        dtyp             real
```

```
Storage Type     styp        dens_full
Row Names        rnam                0
Column Names     cnam                0
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow              175
Number Columns   ncol                2
Lower Bandwidth  lbw               174
Upper Bandwidth  ubw                 1
Size in Bytes    size             3056
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer              302
Smallest Value   vmin                0
Largest Value    vmax           0.6478
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
---------------------------------------
```

```
Matrix D:
 D |         1         2
-----------------------
 1 |   15.6009         0
 2 |         0   12.3752
```

ACLS without truncation:

```
  optn = [ "print"          2 ,      /* ipri */
           "meth"      "acls" ,      /* imet=ACLS */
           "maxit"        400 ,      /* maxiter */
           "tol"        1.e-4 ,      /* term */
           "spar"       1.e-4 ,      /* spar: this must be small */
           "seed"        123 ];      /* seed */
  < w,h > = nnmf(datc,nf,optn);
  attrib(w);
  attrib(h);
```

```
                Nonnegative Matrix Factorization
                Alternative Least Squares Method
```

Iter	OptCrit	DiffCrit	L1Crit	LooCrit
1	2461.89916	276.815934	9816.15639	3.01532613
2	2390.43820	71.4609568	8785.28319	3.03564929
3	2364.27423	26.1639659	8459.00126	3.03076252
4	2352.22336	12.0508708	8330.23142	2.96159867
5	2345.36283	6.86053300	8287.04490	2.90107585
6	2341.96955	3.39327701	8248.40671	2.87550062
7	2340.23145	1.73810580	8223.27913	2.86860053
8	2339.19707	1.03437503	8208.82498	2.87004397
9	2338.50454	0.69252979	8202.46055	2.87425283
10	2337.95241	0.55213533	8199.19539	2.87867176
11	2337.35605	0.59636163	8198.83599	2.88354620
12	2336.63021	0.72583512	8198.95176	2.88952845
13	2335.72687	0.90333690	8198.52711	2.89694474
14	2334.49971	1.22715853	8210.69370	2.90586859
15	2332.56941	1.93030117	8215.84728	2.91719828
16	2329.65240	2.91701475	8207.92530	2.93251552
17	2325.75753	3.89486777	8179.43846	2.95199137
18	2321.59372	4.16380676	8135.93951	2.97370423
19	2318.99580	2.59792115	8110.53947	2.99338962
20	2318.41198	0.58382117	8068.93462	3.00719087
21	2318.26821	0.14377063	8041.67485	3.01528877
22	2318.23132	0.03689018	8025.20145	3.02054860
23	2318.22305	0.00827530	8015.30108	3.02422396
24	2318.22410	-0.00105130	8009.07409	3.02693317

L2 Precision : 2318.22 L1 Precision : 8009.07
Computer Time : 3 seconds

Table of Attributes

Object Name	name	w
Object Type	otyp	matrix_low
Data Type	dtyp	real
Storage Type	styp	spar_packed
Row Names	rnam	0
Column Names	cnam	0
Row Labels	rlab	0
Column Labels	clab	0
Number Rows	nrow	5871
Number Columns	ncol	2
Lower Bandwidth	lbw	5870

Upper Bandwidth	ubw	0
Size in Bytes	size	62236
String Length	slen	0
Number Strings	nstr	0
Number MissVals	nmis	0
Number NonzeroV	nzer	2406
Smallest Value	vmin	0
Largest Value	vmax	3.32
Frobenius Norm	nrm2	.
Recip Condition	rcond	.
Determinant	det	.
Largest SingVal	svb	.
Smallest SingV	svs	.
Num Rank Estim	rnk	.

Table of Attributes

Object Name	name	h
Object Type	otyp	matrix_low
Data Type	dtyp	real
Storage Type	styp	dens_full
Row Names	rnam	0
Column Names	cnam	0
Row Labels	rlab	0
Column Labels	clab	0
Number Rows	nrow	175
Number Columns	ncol	2
Lower Bandwidth	lbw	174
Upper Bandwidth	ubw	0
Size in Bytes	size	3056
String Length	slen	0
Number Strings	nstr	0
Number MissVals	nmis	0
Number NonzeroV	nzer	265
Smallest Value	vmin	0
Largest Value	vmax	0.4465
Frobenius Norm	nrm2	.
Recip Condition	rcond	.
Determinant	det	.
Largest SingVal	svb	.
Smallest SingV	svs	.
Num Rank Estim	rnk	.

ACLS with truncation shows the problems in the last iterations:

```
optn = [ "print"          2 ,      /* ipri */
         "meth"      "acls" ,      /* imet=ACLS */
         "reduc"            ,      /* ivers: red stepwise */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"        123 ];      /* seed */
< w,h > = nnmf(datc,nf,optn);
attrib(w);
attrib(h);
```

```
                Nonnegative Matrix Factorization
                Alternative Least Squares Method

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  2461.89916  276.815934  9816.15639  3.01532613
          2  2390.43820  71.4609568  8785.28319  3.03564929
          3  2364.27423  26.1639659  8459.00126  3.03076252
          4  2352.22336  12.0508708  8330.23142  2.96159867
          5  2345.36283  6.86053300  8287.04490  2.90107585
          6  2341.96955  3.39327701  8248.40671  2.87550062
          7  2340.23145  1.73810580  8223.27913  2.86860053
          8  2339.19707  1.03437503  8208.82498  2.87004397
          9  2338.50454  0.69252979  8202.46055  2.87425283
         10  2337.95241  0.55213533  8199.19539  2.87867176
         11  2337.35605  0.59636163  8198.83599  2.88354620
         12  2336.63021  0.72583512  8198.95176  2.88952845
         13  2335.72687  0.90333690  8198.52711  2.89694474
         14  2334.49971  1.22715853  8210.69370  2.90586859
         15  2332.56941  1.93030117  8215.84728  2.91719828
         16  2329.65240  2.91701475  8207.92530  2.93251552
         17  2325.75753  3.89486777  8179.43846  2.95199137
         18  2321.59372  4.16380676  8135.93951  2.97370423
         19  2318.99580  2.59792115  8110.53947  2.99338962
         20  2318.41198  0.58382117  8068.93462  3.00719087
         21  2318.26821  0.14377063  8041.67485  3.01528877
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
         22  2318.23132  0.03689262  8025.24555  3.02054864
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
         23  2318.22304  0.00827631  8015.34316  3.02422409
          [H]: Reduce Sparsity Bound: from 0.0001 to 0
```

```
          [W]: Reduce Sparsity Bound: from 0.0001 to 0
         24  2318.22409 -0.00105149  8009.12609  3.02693329

        L2 Precision : 2318.22   L1 Precision : 8009.13
                  Computer Time : 10 seconds
```

```
optn = [ "print"          2 ,      /* ipri */
         "meth"      "gcls" ,      /* imet=GCLS */
         "lambda"     1.e-2 ,      /* lambda */
         "maxit"        400 ,      /* maxiter */
         "tol"        1.e-4 ,      /* term */
         "spar"       1.e-4 ,      /* spar: this must be small */
         "seed"           5 ];     /* seed */
< w,h > = nnmf(datc,nf,optn);
attrib(w);
attrib(h);
```

```
              Nonnegative Matrix Factorization
             Gradient Descent with Constrained LS

       Iter     OptCrit    DiffCrit      L1Crit     LooCrit

          1  2440.73154  297.983552  8462.04081  3.00738155
          2  2433.53145  7.20008452  8343.97958  3.00347806
          3  2424.21393  9.31752635  8324.87148  2.98260011
          4  2405.99437  18.2195582  8339.87583  2.95167820
          5  2379.71577  26.2785973  8334.58018  2.93124268
          6  2356.33433  23.3814380  8255.39123  2.94533030
          7  2339.78218  16.5521529  8143.12310  2.98083248
          8  2331.03057  8.75160884  8061.71635  3.01247299
          9  2326.88853  4.14204320  8024.92196  3.03193655
         10  2324.73064  2.15788445  8013.38731  3.04259327
         11  2323.49605  1.23459458  8013.88413  3.04823827
         12  2322.69831  0.79774475  8017.85648  3.05107693
         13  2322.11624  0.58206438  8021.65325  3.05244728
         14  2321.66414  0.45209864  8024.54042  3.05311495
         15  2321.30278  0.36135758  8026.54064  3.05336717
         16  2321.00860  0.29418043  8027.95587  3.05325905
         17  2320.76589  0.24271865  8029.22454  3.05281042
         18  2320.56184  0.20404154  8030.58153  3.05207459
```

```
 19  2320.38840  0.17344838  8032.29333  3.05111900
 20  2320.24184  0.14656075  8034.01559  3.05000726


 21  2320.11404  0.12779214  8035.77376  3.04877797
 22  2320.00185  0.11218832  8037.42318  3.04746487
 23  2319.90318  0.09867256  8039.01062  3.04609810
 24  2319.81634  0.08684224  8040.43510  3.04470165
 25  2319.74014  0.07619794  8041.65195  3.04329541
 26  2319.68117  0.05896841  8042.87371  3.04191415
 27  2319.63351  0.04766418  8043.99357  3.04056809
 28  2319.59460  0.03890473  8045.04764  3.03925816
 29  2319.56288  0.03171972  8046.04210  3.03798837
 30  2319.53733  0.02555166  8046.92852  3.03676397
 31  2319.51889  0.01844035  8047.77771  3.03559254
 32  2319.50802  0.01087454  8048.50283  3.03448003
 33  2319.50140  0.00661585  8049.20981  3.03342392
 34  2319.49837  0.00303033  8049.85928  3.03242160
 35  2319.49821  1.603e-004  8050.46416  3.03147265
 36  2319.50029 -0.00207774  8050.97301  3.03057706

L2 Precision : 2319.5   L1 Precision : 8050.97
          Computer Time : 4 seconds


--------------------------------------
         Table of Attributes
--------------------------------------
Object Name      name               w
Object Type      otyp       matrix_low
Data Type        dtyp             real
Storage Type     styp      spar_packed
Row Names        rnam                0
Column Names     cnam                0
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow             5871
Number Columns   ncol                2
Lower Bandwidth  lbw              5870
Upper Bandwidth  ubw                 0
Size in Bytes    size            63068
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer             2458
Smallest Value   vmin                0
Largest Value    vmax            3.188
```

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


--------------------------------------
         Table of Attributes
--------------------------------------
Object Name      name                h
Object Type      otyp       matrix_low
Data Type        dtyp             real
Storage Type     styp        dens_full
Row Names        rnam                0
Column Names     cnam                0
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow              175
Number Columns   ncol                2
Lower Bandwidth  lbw               174
Upper Bandwidth  ubw                 0
Size in Bytes    size             3056
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer              306
Smallest Value   vmin                0
Largest Value    vmax           0.4293
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
--------------------------------------
```

The number of nonzeros in both matrices compared to those of the former solution shows that the two solutions are rather different. Even though the optimal function value at the local minimas is very close.

```
optn = [ "print"          2 ,      /* ipri */
         "meth"     "macls" ,      /* imet=MACLS */
```

```
        "maxit"        400 ,      /* maxiter */
        "tol"        1.e-4 ,      /* term */
        "spar"       1.e-4 ,      /* spar: this must be small */
        "seed"          5 ];      /* seed */
< w,h,d > = nnmf(datc,nf,optn);
attrib(w);
attrib(h);
/* print "Matrix W:", w; */
print "Matrix H:", h;
print "Matrix D:", d;
```

Nonnegative Matrix Factorization
Modified Alternative Least Squares Method

	Iter	OptCrit	DiffCrit	L1Crit	LooCrit
	1	2458.04983	280.665263	9572.46923	2.85797456
	2	2376.49262	81.5572035	8719.69499	2.75575957
	3	2340.92439	35.5682315	8341.94190	2.83310526
	4	2328.07836	12.8460362	8116.71342	2.93199519
	5	2324.40586	3.67249827	7999.05861	2.98622715
	6	2323.40277	1.00308356	7952.08272	3.01217387
	7	2323.11380	0.28897006	7934.49382	3.02480632
*	8	2322.54217	0.57163015	7930.04944	3.02664214
*	9	2322.42379	0.11838240	7945.50413	3.02444479
*	10	2322.39038	0.03341323	7950.66698	3.02359218
*	11	2322.37540	0.01498285	7952.43815	3.02352718
*	12	2322.36543	0.00996559	7952.98363	3.02389873
*	13	2322.35650	0.00892936	7953.03752	3.02451379
*	14	2322.34698	0.00951782	7952.87697	3.02527505
*	15	2322.33595	0.01103033	7952.60597	3.02613854
*	16	2322.32268	0.01327053	7952.26919	3.02708806
*	17	2322.30644	0.01623724	7951.89800	3.02812596
*	18	2322.28644	0.02000086	7951.48750	3.02925791
*	19	2322.26177	0.02467791	7951.02563	3.03049272
*	20	2322.23129	0.03047313	7950.52955	3.03183933
*	21	2322.19363	0.03765928	7949.99918	3.03330643
*	22	2322.14710	0.04653743	7949.49118	3.03489886
*	23	2322.08984	0.05725974	7948.95247	3.03662178
*	24	2322.01977	0.07006713	7948.47373	3.03847490
*	25	2321.93427	0.08550191	7948.01722	3.04044393
*	26	2321.82926	0.10500964	7947.64419	3.04250914
*	27	2321.70003	0.12923082	7947.41085	3.04466101
*	28	2321.54144	0.15858707	7947.16448	3.04687252

```
*    29  2321.34900  0.19244126  7947.11264  3.04908730
*    30  2321.11968  0.22931654  7947.48886  3.05121905
*    31  2320.85329  0.26639544  7948.86826  3.05315443
*    32  2320.55351  0.29977645  7950.58331  3.05475603
*    33  2320.22956  0.32394892  7953.37525  3.05587477
*    34  2319.89059  0.33897575  7956.55180  3.05638765
*    35  2319.55003  0.34055815  7959.60011  3.05621226
*    36  2319.22721  0.32281556  7962.97722  3.05530393
*    37  2318.93605  0.29116641  7966.83763  3.05374781
*    38  2318.68707  0.24897676  7970.37141  3.05168372
*    39  2318.48448  0.20258588  7973.75837  3.04926775
*    40  2318.32652  0.15796131  7977.38690  3.04667203

*    41  2318.20786  0.11866010  7980.62350  3.04403476
*    42  2318.12191  0.08594675  7983.53048  3.04148288
*    43  2318.06155  0.06035982  7986.17587  3.03912828
*    44  2318.02050  0.04105810  7988.39383  3.03702016
*    45  2317.99294  0.02756108  7990.23082  3.03517750
*    46  2317.97463  0.01830728  7991.72741  3.03359709
*    47  2317.96258  0.01205139  7992.87810  3.03226203
*    48  2317.95474  0.00783890  7993.80420  3.03114668
*    49  2317.94969  0.00504893  7994.54547  3.03022353
*    50  2317.94647  0.00322318  7995.14312  3.02946692
*    51  2317.94440  0.00206064  7995.62088  3.02885297
*    52  2317.94309  0.00131799  7995.99596  3.02835742
*    53  2317.94224  8.426e-004  7996.29857  3.02795845
*    54  2317.94171  5.386e-004  7996.54807  3.02763791
*    55  2317.94136  3.420e-004  7996.75197  3.02738065
*    56  2317.94115  2.168e-004  7996.91327  3.02717454
*    57  2317.94101  1.374e-004  7997.04043  3.02700967
*    58  2317.94092  8.697e-005  7997.14112  3.02687797

   L2 Precision : 2317.94   L1 Precision : 7997.14
            Computer Time : 72 seconds


   ---------------------------------------
            Table of Attributes
   ---------------------------------------
   Object Name      name                w
   Object Type      otyp       matrix_low
   Data Type        dtyp             real
   Storage Type     styp      spar_packed
   Row Names        rnam                0
   Column Names     cnam                0
   Row Labels       rlab                0
```

Column Labels	clab	0
Number Rows	nrow	5871
Number Columns	ncol	2
Lower Bandwidth	lbw	5870
Upper Bandwidth	ubw	0
Size in Bytes	size	62220
String Length	slen	0
Number Strings	nstr	0
Number MissVals	nmis	0
Number NonzeroV	nzer	2405
Smallest Value	vmin	0
Largest Value	vmax	3.229
Frobenius Norm	nrm2	.
Recip Condition	rcond	.
Determinant	det	.
Largest SingVal	svb	.
Smallest SingV	svs	.
Num Rank Estim	rnk	.

Table of Attributes

Object Name	name	h
Object Type	otyp	matrix_low
Data Type	dtyp	real
Storage Type	styp	dens_full
Row Names	rnam	0
Column Names	cnam	0
Row Labels	rlab	0
Column Labels	clab	0
Number Rows	nrow	175
Number Columns	ncol	2
Lower Bandwidth	lbw	174
Upper Bandwidth	ubw	0
Size in Bytes	size	3056
String Length	slen	0
Number Strings	nstr	0
Number MissVals	nmis	0
Number NonzeroV	nzer	268
Smallest Value	vmin	0
Largest Value	vmax	0.4326
Frobenius Norm	nrm2	.
Recip Condition	rcond	.
Determinant	det	.

```
         Largest SingVal  svb                .
         Smallest SingV   svs                .
         Num Rank Estim   rnk                .
         ---------------------------------------
```

```
Matrix D:
 D |         1         2
-----------------------
 1 |   15.0478         0
 2 |         0   13.1772
```

```
  optn = [ "print"          2 ,      /* ipri */
           "meth"     "mgcls" ,      /* imet=MGCLS */
           "lambda"     1.e-2 ,      /* lambda */
           "maxit"        400 ,      /* maxiter */
           "tol"        1.e-4 ,      /* term */
           "spar"       1.e-4 ,      /* spar: this must be small */
           "seed"           5 ];     /* seed */
  < w,h > = nnmf(datc,nf,optn);
  attrib(w);
  attrib(h);
```

```
                Nonnegative Matrix Factorization
            Modified Gradient Descent with Constrained LS

          Iter     OptCrit    DiffCrit      L1Crit     LooCrit

             1  2440.73154  297.983552  8462.04081  3.00738155
      *      2  2393.54133  47.1902069  8511.55448  2.93856303
      *      3  2342.69124  50.8500966  8315.55327  2.90894565
      *      4  2328.17326  14.5179795  8088.47486  2.95653532
      *      5  2324.16315  4.01010444  7991.87882  2.99469943
      *      6  2322.71565  1.44749699  7960.58242  3.01692594
      *      7  2322.03433  0.68132426  7949.99902  3.03037731
      *      8  2321.61313  0.42119506  7946.28421  3.03910190
      *      9  2321.28138  0.33175043  7945.14351  3.04515169
      *     10  2320.97018  0.31120127  7945.60444  3.04950643
      *     11  2320.65150  0.31868215  7947.20215  3.05264053
      *     12  2320.31850  0.33300157  7950.12074  3.05475165
      *     13  2319.97525  0.34325095  7953.95975  3.05593095
      *     14  2319.62991  0.34533507  7957.50595  3.05620165
      *     15  2319.30024  0.32967201  7961.31027  3.05560477
      *     16  2319.00023  0.30001305  7965.35043  3.05424175
      *     17  2318.74082  0.25940398  7969.20497  3.05229777
```

```
*    18  2318.52758  0.21324580  7972.62528  3.04993783
*    19  2318.35957  0.16800931  7976.39363  3.04735578
*    20  2318.23232  0.12724983  7979.76350  3.04469935

*    21  2318.13947  0.09284784  7982.80217  3.04209951
*    22  2318.07370  0.06577284  7985.54650  3.03968233
*    23  2318.02870  0.04500312  7987.90880  3.03750515
*    24  2317.99841  0.03028111  7989.81542  3.03559424
*    25  2317.97826  0.02015919  7991.40770  3.03394928
*    26  2317.96496  0.01330003  7992.62901  3.03255678
*    27  2317.95628  0.00867704  7993.60520  3.03139105
*    28  2317.95068  0.00560157  7994.38636  3.03042470
*    29  2317.94710  0.00358118  7995.01348  3.02963092
*    30  2317.94481  0.00228717  7995.51895  3.02898539
*    31  2317.94334  0.00146359  7995.91646  3.02846417
*    32  2317.94241  9.355e-004  7996.23241  3.02804426
*    33  2317.94181  5.984e-004  7996.49344  3.02770683
*    34  2317.94143  3.804e-004  7996.70841  3.02743588
*    35  2317.94119  2.412e-004  7996.87892  3.02721875
*    36  2317.94104  1.528e-004  7997.01333  3.02704500
*    37  2317.94094  9.679e-005  7997.11956  3.02690618

   L2 Precision : 2317.94   L1 Precision : 7997.12
            Computer Time : 51 seconds

   ------------------------------------
           Table of Attributes
   ------------------------------------
   Object Name      name               w
   Object Type      otyp       matrix_low
   Data Type        dtyp             real
   Storage Type     styp      spar_packed
   Row Names        rnam                0
   Column Names     cnam                0
   Row Labels       rlab                0
   Column Labels    clab                0
   Number Rows      nrow             5871
   Number Columns   ncol                2
   Lower Bandwidth  lbw              5870
   Upper Bandwidth  ubw                 0
   Size in Bytes    size            62204
   String Length    slen                0
   Number Strings   nstr                0
   Number MissVals  nmis                0
   Number NonzeroV  nzer             2404
```

```
Smallest Value   vmin                0
Largest Value    vmax            3.229
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
---------------------------------------


---------------------------------------
         Table of Attributes
---------------------------------------
Object Name      name                h
Object Type      otyp       matrix_low
Data Type        dtyp             real
Storage Type     styp        dens_full
Row Names        rnam                0
Column Names     cnam                0
Row Labels       rlab                0
Column Labels    clab                0
Number Rows      nrow              175
Number Columns   ncol                2
Lower Bandwidth  lbw               174
Upper Bandwidth  ubw                 0
Size in Bytes    size             3056
String Length    slen                0
Number Strings   nstr                0
Number MissVals  nmis                0
Number NonzeroV  nzer              268
Smallest Value   vmin                0
Largest Value    vmax           0.4326
Frobenius Norm   nrm2                .
Recip Condition  rcond               .
Determinant      det                 .
Largest SingVal  svb                 .
Smallest SingV   svs                 .
Num Rank Estim   rnk                 .
---------------------------------------
```

3.6 Function `permute`

nxtprm = permute(sopt,n,curind<,upprng>)

Purpose: The `permute` function computes

- either a specified number of combinations
- or a specified number of permutations

which are followups from a specified starting combination or permutation in `curind`.

Input: **sopt** The first input argument is a string option specifying whether combinations or permutations should be computed. There are only two valid string options:

"comb" combinations are returned

"perm" permutations are returned

n The second input argument n specifies the number of permutations or combinations which should be returned. Specifying a missing value or an integer ≤ 0 means that all possible combinations and permutations are returned within the rows of a matrix. In that case the content of the third argument `curind` is neglected, i.e. permutations will always start with `[ 1 2 .... m ]` and combinations with `[ 1 1 ... 1 ]`. However, the dimension of the `curind` vector is always significant.

curind The third input argument is a vector with m int values specifying the current (starting) combination or permutation. If all combinations should be computed, the specification of `curind = [ 1 1 ... 1 ]`, and if all permutations should be computed the specification of `curind = [ 1 2 ... m ]` would be appropriate.

upprng This input is valid only for combinations. The fourth input argument may be an int scalar scalar or a vector of m integers specifying the upper range of ints in the combinations. Note, that the dimension of the vector `upprng` must agree with the dimension of the `curind` vector. When a scalar is specified it will is used in the same way as a vector with all entries set to this scalar value.

Output: Only one output argument is returned which is either a matrix or a vector of integers containing either a specified number of followup combinations or permutations or simply the next combination or permutation.

Restrictions: 1. No missing values are permitted for `sopt` and `curind`. For combinations, `upprng` cannot be missing.

2. The dimension of `curind` may be too large to compute all the combinations or permutations. If `upprng` is a vector its dimension must agree with that of `curind`.

Relationships:

Examples: 1. Combinations (with replication):

```
   curind = [ 1 1 ];
   uprng = 3;
   allcom = permute("comb",.,curind,uprng);
   print "1. all comb=", allcom;
```

```
1. all comb=
   |   1   2
------------
 1 |   1   1
 2 |   1   2
 3 |   1   3
 4 |   2   1
 5 |   2   2
 6 |   2   3
 7 |   3   1
 8 |   3   2
 9 |   3   3
```

```
   nn = 3 * 3;
   uprng = 3;
   allcom = curind = [ 1 1 ];
   for (i = 2; i <= nn; i++) {
     curind = permute("comb",1,curind,uprng);
     allcom = allcom |> curind;
   }
   print "2. stepwise all comb=", allcom;
```

```
2. stepwise all comb=
   |   1   2
------------
 1 |   1   1
 2 |   1   2
 3 |   1   3
 4 |   2   1
 5 |   2   2
 6 |   2   3
 7 |   3   1
 8 |   3   2
 9 |   3   3
```

```
   curind = [ 2 2 ];
   uprng  = [ 3 4 ];
   allcom = permute("comb",100,curind,uprng);
```

```
   print "3. more advanced: comb=", allcom;


3. more advanced: comb=
   |   1   2
------------
 1 |   2   3
 2 |   2   4
 3 |   3   1
 4 |   3   2
 5 |   3   3
 6 |   3   4


   curind = [ 2 2 ];
   uprng  = [ 3 4 ];
   nxtcom = permute("comb",1,curind,uprng);
   print "4. single nxtcomb: comb=", nxtcom;


4. single nxtcomb: comb=
   |   1   2
------------
 1 |   2   3
```

2. Permutations (without replication):

```
   curind = [ 1 2 3 ];
   allprm = permute("perm",.,curind);
   print "1. all perm=", allprm;


1. all perm=
   |   1   2   3
---------------
 1 |   1   2   3
 2 |   1   3   2
 3 |   2   1   3
 4 |   2   3   1
 5 |   3   1   2
 6 |   3   2   1


   nn = 2 * 3;
   allcom = curind = [ 1 2 3 ];
   for (i = 2; i <= nn; i++) {
     curind = permute("perm",1,curind);
```

```
    allcom = allcom |> curind;
  }
  print "2. stepwise all perm=", allcom;
```

```
2. stepwise all perm=
   |   1   2   3
---------------
 1 |   1   2   3
 2 |   1   3   2
 3 |   2   1   3
 4 |   2   3   1
 5 |   3   1   2
 6 |   3   2   1
```

```
  curind = [ 2 1 3 ];
  allprm = permute("perm",5,curind);
  print "3. some perm=", allprm;
```

```
3. some perm=
   |   1   2   3
---------------
 1 |   2   3   1
 2 |   3   1   2
 3 |   3   2   1
```

3.7 Function `quantile`

quant = quantile(a,k)

Purpose: The `quantile` function computes k quantiles for each of the n columns of the $m \times n$ input data matrix `A`.

Input: **a** The first input argument specifies a $m \times n$ data matrix.

k The integer k specifies the quantile, i.e. $k = 4$ for quartiles or $k = 10$ for percentiles.

Output: The result is a $n \times k+1$ matrix containing the $k+1$ quantiles (including minimum and maximum) for each of the n columns.

Restrictions: 1. Missing values in input data `a` are not counted for quantiles.

2. No complex or string data can be used.

Relationships: univar(), histogrm()

Examples: 1. Heart data, D.M. Hawkins (1994)

```
a= [ 1 42.8 40.0 37,
     2 63.5 93.5 50,
     3 37.5 35.5 34,
     4 39.5 30.0 36,
     5 45.5 52.0 43,
     6 38.5 17.0 28,
     7 43.0 38.5 37,
     8 22.5  8.5 20,
     9 37.0 33.0 34,
    10 23.5  9.5 30,
    11 33.0 21.0 38,
    12 58.0 79.0 47 ];


   aa = a[,2:4];
   nr = nrow(aa); nc = ncol(aa);
   sopt= [ "ari" "qua" ];
   c1 = univar(aa,sopt);
   print "\n Univar Result: Quartiles\n",c1;
```

```
 Univar Result: Quartiles

           |     Var_1     Var_2     Var_3
-------------------------------------------
  Ari_Mean |   40.3583   38.1250   36.1667
  Median   |   39.0000   34.2500   36.5000
  Quart_1  |   35.0000   19.0000   32.0000
  Quart_3  |   44.2500   46.0000   40.5000
```

```
   quant = quantile(aa,4);
   print "Quartiles=", quant;
```

The quantile matrix contains the minimum and maximum in its first and last column:

```
Quartiles=
   |         1         2         3         4         5
-------------------------------------------------------
 1 |   22.5000   35.0000   39.0000   44.2500   63.5000
 2 |    8.5000   19.0000   34.2500   46.0000   93.5000
 3 |   20.0000   32.0000   36.5000   40.5000   50.0000
```

3.8 Function `simdid`

dist = simdid(a,optn)

Purpose: The `simdid` function computes a $n \times n$ matrix **S** of similarity (distance) measures among n histograms (frequencies of discrete distributions) with p values provided in the n columns of the $m \times n$ input data matrix `A`. If **A** has only $n = 2$ columns, then only a scalar (and not a 2×2 matrix) is returned.

Input: a The first input argument specifies a $m \times n$ data matrix and must not have string or missing values. Note, the data values must be in $[0, 1)$ and the values of each column must add to one.

optn Specifies an options vector with the following content:

1. an integer specifying the amount of printed output.
2. an integer specifying the type of measure:
   - **1** Hellinger similarity coefficient
   - **2** Chernoff similarity coefficient
   - **3** Jeffreys (dissimilarity) distance
   - **4** Directed Divergence distance (Kullback, 1978)
   - **5** Symmetric Divergence distance (Kullback, 1978)
   - **6** Cosine Divergence distance (Chung et al, 1989)
3. parameter α for imet=2 and imet=6
4. set zero columns to small value
5. integer != 0: melting zero columns with neighbor

Output: The result is either a scalar (for $n = 2$) or a $n \times n$ similarity matrix (for $n > 2$) containing the $k+1$ quantiles (including minimum and maximum) for each of the n columns.

Restrictions:

1. There should be no string or missing values in input data `a`.
2. The data values should be in $[0, 1)$ and must add to unity in each column.

Relationships: univar(), histogrm()

Examples:

1. Heart data, D.M. Hawkins (1994)

```
a= [ 1 42.8 40.0 37,
     2 63.5 93.5 50,
     3 37.5 35.5 34,
     4 39.5 30.0 36,
     5 45.5 52.0 43,
     6 38.5 17.0 28,
```

```
      7 43.0 38.5 37,
      8 22.5  8.5 20,
      9 37.0 33.0 34,
     10 23.5  9.5 30,
     11 33.0 21.0 38,
     12 58.0 79.0 47 ];


    aa = a[,2:4];
    nr = nrow(aa); nc = ncol(aa);
    optn = [ 2 ,    /* ipri */
             1 ];   /* same */
    hist = histogrm(aa,4,optn);
    print "Histogram(same=1)=", hist;
    prob = hist‘ / (real)nr;
    print "Probabilities=", prob;


Histogram(same=1)=
   |    1    2    3    4
-------------------------
 1 |    2    8    2    0
 2 |    4    5    1    2
 3 |    2   10    0    0


Probabilities=
   |         1         2         3
-----------------------------------
 1 |   0.16667   0.33333   0.16667
 2 |   0.66667   0.41667   0.83333
 3 |   0.16667   0.08333   0.00000
 4 |   0.00000   0.16667   0.00000


   optn = [ 2 ,    /* ipri */
            1 ,    /* imet=Hellinger */
           .5 ,    /* alfa */
         1.e-6 ];   /* rnu */
   dist = simdidi(prob,optn);
   print "Hellinger Similarity=", dist;


Hellinger Similarity=
 S |         1         2         3
-----------------------------------
 1 |   1.00000
 2 |   0.88101   1.00000
```

```
 3 |   0.91243   0.82565   1.00000
```

```
   /* for alpha=.5: Chernoff is the same as Hellinger */
   optn = [ 2 ,    /* ipri */
            2 ,    /* imet=Chernoff */
           .5 ,    /* alfa */
        1.e-6 ];   /* rnu */
   dist = simdidi(prob,optn);
   print "Chernoff Similarity=", dist;
```

```
Chernoff Similarity=
 S |         1         2         3
-----------------------------------
 1 |   1.00000
 2 |   0.88101   1.00000
 3 |   0.91243   0.82565   1.00000
```

```
   optn = [ 2 ,    /* ipri */
            3 ,    /* imet=Jeffrey */
           .5 ,    /* alfa */
        1.e-6 ];   /* rnu */
   dist = simdidi(prob,optn);
   print "Jeffrey Distance =", dist;
```

```
Jeffrey Distance =
 S |         1         2         3
-----------------------------------
 1 |   0.00000
 2 |   0.23799   0.00000
 3 |   0.17514   0.34869   0.00000
```

```
   optn = [ 2 ,    /* ipri */
            4 ,    /* imet=DirectedDivergence */
           .5 ,    /* alfa */
        1.e-6 ];   /* rnu */
   dist = simdidi(prob,optn);
   print "Directed Divergence Distance=", dist;
```

The directed divergence is unsymmetric:

```
Directed Divergence Distance=
   |         1         2         3
```

```
---------------------------------
 1 |   0.00000   0.31332   1.85520
 2 |   1.98141   0.00000   2.89041
 3 |   0.18594   0.46207   0.00000


   optn = [ 2 ,    /* ipri */
            5 ,    /* imet=SymmetricDivergence */
           .5 ,    /* alfa */
        1.e-6 ];   /* rnu */
   dist = simdidi(prob,optn);
   print "Symmetric Divergence=", dist;


Symmetric Divergence=
 S |         1         2         3
---------------------------------
 1 |   0.00000
 2 |   2.29473   0.00000
 3 |   2.04114   3.35249   0.00000


   optn = [ 2 ,    /* ipri */
            6 ,    /* imet=CosineDivergence */
           .5 ,    /* alfa */
        1.e-6 ];   /* rnu */
   dist = simdidi(prob,optn);
   print "Symmetric Divergence Distance=", dist;


Symmetric Divergence Distance=
 S |         1         2         3
---------------------------------
 1 |   0.00000
 2 |   0.07726   0.00000
 3 |   0.04611   0.11173   0.00000
```

3.9 Function svdupd

<s2,v2,u2> = svdupd(a,b<,s0,v0,u0>)

Purpose: The svdupd function computes the rank r update of the truncated SVD $(\mathbf{U2}, \mathbf{S2}, \mathbf{V2})$ of an $m \times n$ matrix $\mathbf{X}$

$$(\mathbf{U2}, \mathbf{S2}, \mathbf{V2}) = svd(\mathbf{X} + \mathbf{A}^T\mathbf{B})$$

using the SVD $(\mathbf{U0}, \mathbf{S0}, \mathbf{V0})$ of $\mathbf{X}$.

Input:

Output:

Restrictions: 1.

2.

Relationships:

Examples: 1. Data are: Holtzinger centroid pattern reported by Harman (1960,211): There are $N = 24$ observations and $n = 4$ variables.

```
a = [ 1  608 -116  300 -250 ,
      2  372 -119  207 -135 ,
      3  427 -220  262 -155 ,
      4  477 -211  206 -184 ,
      5  668 -306 -344  108 ,
      6  661 -337 -258  216 ,
      7  652 -396 -384  124 ,
      8  662 -225 -153  -60 ,
      9  664 -394 -240  308 ,
     10  462  455 -365 -136 ,
     11  569  397 -208  -63 ,
     12  484  360 -149 -388 ,
     13  608  130  -99 -402 ,
     14  442  199  -13  293 ,
     15  407  170  146  266 ,
     16  523   77  300   76 ,
     17  492  317   82  338 ,
     18  547  307  248   72 ,
     19  452  125  129  111 ,
     20  612 -174  128    4 ,
     21  601  114   80 -171 ,
     22  608 -144  145  136 ,
     23  691 -164  129 -116 ,
     24  654  151 -150   -3 ];
```

Compute SVD of all data :

```
print "SVD for all data: (s0,v0,u0)";
a0 = a[ ,2:5];
< s0,v0,u0 > = svd(a0);
print "S0=", s0;
print "V0=", v0;
```

```
S0=
```

```
 D |         1         2         3         4
-------------------------------------------
 1 |   2767.14         0         0         0
 2 |         0   1290.42         0         0
 3 |         0         0   1074.55         0
 4 |         0         0         0    951.90


V0=
   |         1         2         3         4
-------------------------------------------
 1 |   0.99838   0.05066   0.01703   0.01975
 2 |  -0.04640   0.93097  -0.29414   0.21125
 3 |  -0.03293   0.19662   0.90220   0.38248
 4 |   0.00298  -0.30343  -0.31500   0.89927
```

Now, compute SVD of the first 23 observations and update for last observation:

```
/* Update only one observation: 24 */
a1 = a[1:23,2:5];
< s1,v1,u1 > = svd(a1,"eco");
b2 = a[24,2:5]‘;
< s2,v2,u2 > = svdupd(.,b2,s1,v1,u1);
print "S2=", s2;
print "V2=", v2;
```

```
S2=

 D |         1         2         3         4
-------------------------------------------
 1 |   2767.14         0         0         0
 2 |         0   1290.42         0         0
 3 |         0         0   1074.55         0
 4 |         0         0         0    951.90


V2=
   |         1         2         3         4
-------------------------------------------
 1 |   0.99838   0.05066   0.01703   0.01975
 2 |  -0.04640   0.93097  -0.29414   0.21125
 3 |  -0.03293   0.19662   0.90220   0.38248
 4 |   0.00298  -0.30343  -0.31500   0.89927
```

Now use SVD of first 23 observations and update for last four observations:

```
/* Update four observations: 21,..,24 */
a1 = a[1:20,2:5];
< s1,v1,u1 > = svd(a1,"eco");
b2 = a[21:24,2:5]‘;
< s2,v2,u2 > = svdupd(.,b2,s1,v1,u1);
print "S2=", s2;
print "V2=", v2;
```

```
S2=
 D |         1         2         3         4
-------------------------------------------
 1 |   2767.14         0         0         0
 2 |         0   1290.42         0         0
 3 |         0         0   1074.55         0
 4 |         0         0         0    951.90
```

```
V2=
   |         1         2         3         4
-------------------------------------------
 1 |   0.99838   0.05066   0.01703   0.01975
 2 |  -0.04640   0.93097  -0.29414   0.21125
 3 |  -0.03293   0.19662   0.90220   0.38248
 4 |   0.00298  -0.30343  -0.31500   0.89927
```

Now use SVD of first 3 variables and update for last variable:

```
/* Update for last variable */
a1 = a[1:24,2:4];
< s1,v1,u1 > = svd(a1,"eco");
a2 = a[ ,5];
< s2,v2,u2 > = svdupd(a2,.,s1,v1,u1);
print "S2=", s2;
print "V2=", v2;
```

4 Illustration

4.1 Bootstrap with the `noharm` Function

I have added naive (observationwise) and parametric bootstrap methods for ASEs and confidence intervals to the `noharm()` function in CMAT. When using the *LSAT6* data set with $N = 1000$ observations and $n = 5$ (binary) variables I can compare the following three results:

1. The Satorra-Bentler robust (non-normal) ASEs and confidence intervals for the estimates of five thresholds and the five loadings of a one factor solution. This method needs the 100×5 raw data set.

2. The naive bootstrap ASEs and confidence intervals for the normal, the bias corrected and the BCa methods. This method needs the 100×5 raw data set.

3. Asymptotic standard errors and confidence intervcals generated using the parametric bootstrap method which generates multivariate normal $\mathcal{N}(\mu, \Sigma)$ random generated samples of $n \times n$ scalar product matrices $\mathbf{S}$ and computes its thresholds and factor loadings.

As we can see, both, the ASEs and CIs of the first two methods (non-normal SB approach and naive bootstrap) are very close. This is probably due to the large sample size. On the other hand the parametric bootstrap creates much larger ASEs and CIs probably due to the fact that the created matrices no longer have some of the properties of scalar product matrices of data sets consisting of binary variables.
The following specifies the input of the *LSAT6* data set from the `tdata` directory:

```
  options NOECHO;
  lsat6 = [
#include "..\\tdata\\lsat6.dat"
           ];
  options ECHO;

  lsat6 = shape(lsat6,.,5);
  name = [ "V1" : "V5" ];
  lsat6 = cname(lsat6,name);
```

1. Using the following specification

```
     optn = [ "data"      "raw" ,
              "nfac"          1 ,
              "cl"       "wald" ,
              "frot"   "promax" ,
              "prin"          3 ];
```

```
< gof,parm,resi,covm > = noharm(lsat6,optn);
```

we show some results of the `noharm()` function leading up to the output of the Satorra-Bentler robust ASEs and CIs:

```
           Constants, Thresholds, and Unique Variances
           *******************************************

           N  Variable   Constants  Thresholds     UniVars

           1         V1 -1.56515972 -1.43250271  0.83767119
           2         V2 -0.59673612 -0.55046572  0.85093405
           3         V3 -0.15143386 -0.13324453  0.77419923
           4         V4 -0.77171121 -0.71598601  0.86079443
           5         V5 -1.19974463 -1.12639118  0.88145644


 Unrotated Factor Loadings
 -------------------------

   1 :       0.4029     0.3861     0.4752     0.3731     0.3443


             Covariance Matrix of Threshold Estimates

               T_1        T_2        T_3        T_4        T_5
    T_1   0.0034345
    T_2   1.81e-004  0.0017552
    T_3   2.30e-004  1.91e-004  0.0015810
    T_4   1.13e-004  1.14e-004  1.89e-004  0.0018971
    T_5   7.01e-005  1.82e-004  1.06e-004  2.17e-004  0.0025273


         Robust Covariance Matrix of Factor Model Estimates

               L_1        L_2        L_3        L_4        L_5
    L_1    0.0124289
    L_2   -7.02e-005  0.0073387
    L_3   -0.0017358 -0.0021169  0.0086842
    L_4   -2.11e-004 -8.84e-004 -0.0017486  0.0075326
    L_5   -3.20e-004 -3.39e-004 -0.0014112  2.25e-004  0.0091070


              Satorra-Bentler Adjusted Residual Matrix
```

```
            V1         V2         V3         V4         V5
 V1      .
 V2  -0.2562815     .
 V3  -0.7463208 -0.4506211     .
 V4   0.6813483  1.2335788 -0.5282240     .
 V5   0.8845747 -0.7929283  1.6546485 -1.3078884     .


     Average Off-diagonal Residual = 8.53641435e-001
                 Sum-of-Squares= 8.95035


            Rank Order of 7 Largest Residuals

     V5,V3     V5,V4     V4,V2     V5,V1     V5,V2     V3,V1
 1.654648 -1.307888  1.233579  0.884575 -0.792928 -0.746321

                          V4,V1
                       0.681348


 Fit criterion . . . . . .     0.0001
 Nobs * OptCrit. . . . . .     0.0947
 Root mean square of residuals .     0.0031
 Tanaka index of goodness of fit .     0.9988
 Satorra-Bentler Mean (df= 5). .  5.0235 Prob>chi**2 = 0.4130
 Sat.-B. M-V (df= 4.323) . . . .  4.3433 Prob>chi**2 = 0.4069
```

The following table shows thresholds and loadings with robust ASEs and CIs:

	Estimate	AStdErr	T_Value	Low_Wald	Upp_Wald
T_1	-1.4325	0.0586	-24.4435	-1.5474	-1.3176
T_2	-0.5505	0.0419	-13.1393	-0.6326	-0.4684
T_3	-0.1332	0.0398	-3.3511	-0.2112	-0.0553
T_4	-0.7160	0.0436	-16.4384	-0.8014	-0.6306
T_5	-1.1264	0.0503	-22.4058	-1.2249	-1.0279
L_1	0.4029	0.1115	3.6139	0.1844	0.6214
L_2	0.3861	0.0857	4.5069	0.2182	0.5540
L_3	0.4752	0.0932	5.0991	0.2925	0.6578
L_4	0.3731	0.0868	4.2989	0.2030	0.5432
L_5	0.3443	0.0954	3.6079	0.1573	0.5313

2. The naive bootstrap techniques specified with

```
optn = [ "data"      "raw" ,
         "nfac"          1 ,
         "cl"       "boot" ,
         "btask"     "bca" ,
         "samp"        500 ,
         "bseed"     12345 ,
         "frot"   "varmax" ,
         "prin"          3 ];
< gof,parm,resi,covm,boci > = noharm(lsat6,optn);
```

   give very similar results:

3. Normal CIs: corrected and uncorrected:

```
        ************************************************
        Bootstrap: ASE, Bias, and Normal CI (500 Samples)
        ************************************************

          Statistics    Estimate     BootASE    BootBias
                 T_1 -1.43250271  0.06201454 -0.00328681
                 T_2 -0.55046572  0.04219817  8.764e-004
                 T_3 -0.13324453  0.04030350 -0.00147406
                 T_4 -0.71598601  0.04365677 -0.00194468
                 T_5 -1.12639118  0.04934390  5.134e-005
                 L_1  0.40290050  0.10889404 -0.02009500
                 L_2  0.38609060  0.09397694  0.00707047
                 L_3  0.47518498  0.11601861  0.01082562
                 L_4  0.37310262  0.09627068  0.00465378
                 L_5  0.34430156  0.11035740  0.00515400


    Statistics     U.N_LCI     U.N_UCI     B.N_LCI     B.N_UCI
           T_1 -1.55076217 -1.30766963 -1.55404898 -1.31095644
           T_2 -0.63404899 -0.46863522 -0.63317260 -0.46775883
           T_3 -0.21076386 -0.05277706 -0.21223793 -0.05425113
           T_4 -0.79960703 -0.62847563 -0.80155171 -0.63042031
           T_5 -1.22315478 -1.02973026 -1.22310345 -1.02967892
           L_1  0.20956712  0.63642389  0.18947211  0.61632889
           L_2  0.19482871  0.56321154  0.20189918  0.57028201
           L_3  0.23696707  0.69175166  0.24779269  0.70257728
           L_4  0.17976177  0.55713591  0.18441555  0.56178969
           L_5  0.12285102  0.55544410  0.12800502  0.56059809
```

4. Bias Corrected BC method:

	Estimate	BOOT_ASE	BOOT_BIAS
T_1	-1.43250	0.06201	-0.00329
T_2	-0.55047	0.04220	0.00088
T_3	-0.13324	0.04030	-0.00147
T_4	-0.71599	0.04366	-0.00194
T_5	-1.12639	0.04934	0.00005
L_1	0.40290	0.10889	-0.02010
L_2	0.38609	0.09398	0.00707
L_3	0.47518	0.11602	0.01083
L_4	0.37310	0.09627	0.00465
L_5	0.34430	0.11036	0.00515

	BC_LCI	BC_UCI
T_1	-1.55477	-1.31355
T_2	-0.64181	-0.47330
T_3	-0.21342	-0.05643
T_4	-0.79950	-0.63412
T_5	-1.22653	-1.03429
L_1	0.21965	0.66525
L_2	0.16349	0.55012
L_3	0.28283	0.69328
L_4	0.15025	0.54673
L_5	0.09101	0.52197

5. Bias Corrected BCa method (skewness corrected):

	Estimate	BOOT_ASE	BOOT_BIAS
T_1	-1.43250	0.06201	-0.00329
T_2	-0.55047	0.04220	0.00088
T_3	-0.13324	0.04030	-0.00147
T_4	-0.71599	0.04366	-0.00194
T_5	-1.12639	0.04934	0.00005
L_1	0.40290	0.10883	-0.02013
L_2	0.38609	0.09400	0.00708
L_3	0.47518	0.11602	0.01083
L_4	0.37310	0.09627	0.00465
L_5	0.34430	0.11036	0.00515

	BCA_LCI	BCA_UCI
T_1	-1.55477	-1.31058
T_2	-0.64027	-0.47330
T_3	-0.21342	-0.05392
T_4	-0.79778	-0.63106
T_5	-1.22123	-1.03215
L_1	0.21872	0.66525
L_2	0.15288	0.54821
L_3	0.28009	0.69230
L_4	0.15025	0.54673
L_5	0.09101	0.51966

4.2 Function BDM: Bivariate Dale Model

A SAS macro for the bivariate Dale model was submitted to JSS (McMillan & Hanson, 2005) and reviewed by the author of CMAT. This model was developed and illustrated in two papers by Dale (1985, 1986). Here the response is a frequency table (f, q) of two ordinal variables f and q with r and c categories. For describing the model, the authors of the macro use the following illustration:

1. Let d be a binary indicator, where $d = 1$ denotes drinking, $d = 0$ denotes abstinence.

2. The two response vars are two ordinal variables measuring the frequency and quantity of alcohol use of respondents in a survey.

3. The probability that an individual with covariate x drinks is modeled with logistic regression:
$$logit\{P(d = 1)\} = x^T \beta_1$$
with a vector of regression coefficients β_1.

4. Conditional on $d = 1$ the bivariate ordinal table (f, q) is modeled using a BDM (Dale, 1986; Molenbergh & Lesaffre, 1994).

5. The marginal probabilities $P(f \leq i), i = 1, \ldots, r-1$ and $P(q \leq j), j = 1, \ldots, c-1$ are modeled using logistic regression:
$$logit\{P(f \leq i)\} = \Theta_{f,i} + x^T \beta_2$$
$$logit\{P(q \leq j)\} = \Theta_{q,j} + x^T \beta_3$$

6. Dependence between f and q is modeled using a *global cross-ratio* (GCR) model, which is a measure of association for contingency tables:
$$\Psi_{i,j} = \frac{P(f \leq i, q \leq j)P(f > i, q > j)}{P(f > i, q \leq j)P(f \leq i, q > j)}$$
for a table dichotomized at cutpoints (i, j).

7. With that the following log-linear model is assumed

$$log(\Psi_{i,j}) = \delta + \alpha_i + \gamma_j + x^T\beta_4$$

where $\alpha_{r-1} = \gamma_{c-1} = 0$ and only $r-2$ parms α_i and $c-2$ parms $gamma_j$ are estimated.

The SAS macro is using the SAS PROCs NLMIXED, GENMOD, LOGISTIC, FREQ and many occasions of the DATA step. The call of the NLMIXED function could have been easily replaced by any call of PROC NLIN or PROC NLP in SAS/OR. There were no problems to write a CMAT function covering almost all features of the SAS Macro.
The following is a CMAT macro that will compute the same results as those of the SAS macro for the examples 1 and 2 in the JSS paper.

```
function BDM(ipri,datb,rowvar,colvar,count,
             rowpred,colpred,gcrpred,cndpred,gcrrow,gcrcol,
             rowmap,colmap,gcrmap,cndmap,mapping) {
nr2 = nrow(datb);
/* rowvar=pain=datb[,1], colvar=med=datb[,2] */
nr0 = datb[<>,rowvar]; nc0 = datb[<>,colvar];
nrm = nr0 - 1; ncm = nc0 - 1;
varnams = cname(datb);
rnam = varnams[rowvar]; cnam = varnams[colvar];


/* only one or no var is permitted currently:
   rowpred,colpred,gcrpred,cndprd must be int or missing */
nrowp = (rowpred == .) ? 0 : datb[<>,rowpred];
ncolp = (colpred == .) ? 0 : datb[<>,colpred];
ngcrp = (gcrpred == .) ? 0 : datb[<>,gcrpred];
ncond = (cndpred == .) ? 0 : datb[<>,cndpred];
nrc = nr0 * nc0; ncc = 4;
free colnam, adold, adcol, advar, adrng, admap;
colnam[1] = rnam;   colnam[2] = cnam;
colnam[3] = "freq"; colnam[4] = "count";
nadc = 0; nrp = ncp = ngp = nco = 1;
rowpr2 = colpr2 = gcrpr2 = cndpr2 = .;
if (nrowp) { nrp = nrowp;
             adcol[++nadc] = rowpr2 = ++ncc;
             adold[nadc] = rowpred; adrng[nadc] = nrowp;
             admap[nadc] = rowmap; nrc *= nrowp;
             colnam[ncc] = varnams[rowpred]; }
if (ncolp) { ncp = ncolp;
             if (colpred == rowpred) colpr2 = rowpr2;
             else { adcol[++nadc] = colpr2 = ++ncc;
                    adold[nadc] = colpred; adrng[nadc] = ncolp;
```

```
                    admap[nadc] = colmap; nrc *= ncolp;
                    colnam[ncc] = varnams[colpred]; } }
if (ngcrp) { ngp = ngcrp;
             if (gcrpred == rowpred) gcrpr2 = rowpr2;
             if (gcrpred == colpred) gcrpr2 = colpr2;
             else { adcol[++nadc] = gcrpr2 = ++ncc;
                    adold[nadc] = gcrpred; adrng[nadc] = ngcrp;
                    admap[nadc] = gcrmap; nrc *= ngcrp;
                    colnam[ncc] = varnams[gcrpred]; } }
if (ncond) { nco = ncond;
             if (cndpred == rowpred) cndpr2 = rowpr2;
             if (cndpred == colpred) cndpr2 = colpr2;
             if (cndpred == gcrpred) cndpr2 = gcrpr2;
             else { adcol[++nadc] = cndpr2 = ++ncc;
                    adold[nadc] = cndpred; adrng[nadc] = ncond;
                    admap[nadc] = cndmap; nrc *= ncond;
                    colnam[ncc] = varnams[cndpred]; } }
if (ipri) {
  print "nrp,ncp,ngp=",nrp,ncp,ngp;
  print "nrc,ncc,nadc=",nrc,ncc,nadc;
  print "rowpr2,colpr2,gcrpr2,cndpr2=",rowpr2,colpr2,gcrpr2,cndpr2;
  print "Colnames=",colnam;
}


/* Create: datc[nrc,ncc]= { ir, ic, freq, sum, <i3,i4,i5> } */
irng = (nadc == 1) ? 1 : cons(nadc,1,1);
comb = permute("comb",.,irng,adrng);
ncmp = nrow(comb);
if (ipri > 1) print "comb=", ncmp, comb;
datc = cons(nrc,ncc,.);
in = 1;
for (kk = 1; kk <= ncmp; kk++) {
  for (ir = 1; ir <= nr0; ir++)
  for (ic = 1; ic <= nc0; ic++, in++) {
    freq = sum = 0;
    for (i = 1; i <= nr2; i++)
    if (datb[i,rowvar] == ir && datb[i,colvar] == ic) {
      if (nadc == 1) {
        if (datb[i,adold] == comb[kk]) { sum += datb[i,count]; freq++; }
      } else {
        for (ii = 1; ii <= nadc; ii++)
        if (datb[i,adold[ii]] != comb[kk,ii]) break;
        if (ii > nadc) { sum += datb[i,count]; freq++; }
    } }
    datc[in,1] = ir;   datc[in,2] = ic;
```

```
    datc[in,3] = freq; datc[in,4] = sum;
    if (nadc == 1) datc[in,5] = mapping[admap+comb[kk]];
    else {
      for (j = 1; j <= nadc; j++)
      datc[in,4+j] = mapping[admap[j]+comb[kk,j]];
} } }
datc = cname(datc,colnam);
if (ipri > 1) print "DATC=",datc;


/* Fit ROWvar: Intercept plus rowpred: but FREQ var= CT */
np   = (nrowp) ? nr0 : nrm;
rmod = (nrowp) ? sprintf("1= %i",rowpr2) : rmod = "1= ";
if (ipri > 1) print "rmod=",rmod;
clas = 1; /* clas = (nrowp) ? [ 1 rowpr2 ] : 1; */
optn = [ "print"          3 ,
         "link"      "logit",
         "dist"      "binom",
         "freq"           4 ,
         "notype1"          ,
         "notype3"          ,
         "tech"     "trureg" ];
< rgof,rowlog > = glim(datc,rmod,optn,clas);

if (ipri) {
  print "Row: gof=", rgof;
  print "Row Likelihood", rlik = rgof[8];
  print "Parms of Logistic=", rowlog;
}
rnam = rname(rowlog);
for (j = 1; j <= np; j++) rnam[j] = sprintf("row_%s",rnam[j]);
rowlog = rname(rowlog,rnam);
if (ipri) print "Row: parm=",rowlog;


/* Fit COLvar: Intercept plus colpred: but FREQ var= CT */
np   = (ncolp) ? nc0 : ncm;
cmod = (ncolp) ? sprintf("2= %i",colpr2) : "2= ";
clas = 2; /* clas = (ncolp) ? [2 colpr2 ] : 2; */
optn = [ "print"          3 ,
         "link"      "logit",
         "dist"      "binom",
         "freq"           4 ,
         "notype1"          ,
         "notype3"          ,
         "tech"     "trureg" ];
```

```
< cgof,collog > = glim(datc,cmod,optn,clas);

if (ipri) {
  print "Col: gof=", cgof;
  print "Col Likelihood", clik = cgof[8];
}
cnam = rname(collog);
for (j = 1; j <= np; j++) cnam[j] = sprintf("col_%s",cnam[j]);
collog = rname(collog,cnam);
if (ipri) print "Col: parm=",collog;

sumfit = 2.*(rlik + clik);
if (ipri) print "Sumfit=",sumfit;
npc = nrow(collog); npr = nrow(rowlog);
parm = collog |> rowlog;
if (ipri) print "Parm=",parm;


/*--- Obtain Starting values:
      Try all 2 by 2 freq tables ---*/
free alltab;
for (ir = 1; ir < nr0; ir++) {
  gentab = cons(ncm,3,0.);
  for (ic = 1; ic < nc0; ic++) {
    ditab = cons(2,2,0.);
    for (ii = 1; ii <= nrc; ii++) {
      j = datc[ii,1]; k = datc[ii,2];
      j = (j <= ir) ? 1 : 2;
      k = (k <= ic) ? 1 : 2;
      ditab[j,k] += datc[ii,4] + 1.;
    }
    if (ipri > 1) print "DITAB=", ditab;
    /* Odds ratio and relative risk of 2 by 2 table */
    den = ditab[1,2] * ditab[2,1];
    odd = ditab[1,1] * ditab[2,2] / den;
    gentab[ic,1] = ir; gentab[ic,2] = ic;
    gentab[ic,3] = odd;
    if (ipri > 1) print "Odds Ratio=",odd;
    sum1 = ditab[1,1] + ditab[1,2];
    sum2 = ditab[2,1] + ditab[2,2];
    p11 = ditab[1,1] / sum1; p12 = ditab[2,1] / sum2;
    rr1 = p11 / p12;
    if (ipri > 1) print "RelRisk1=",rr1;
    p21 = ditab[1,2] / sum1; p22 = ditab[2,2] / sum2;
    rr2 = p21 / p22;
    if (ipri > 1) print "RelRisk2=",rr2;
```

```
  }
  cnam = [ "row" "col" "odds" ];
  gentab = cname(gentab,cnam);
  if (ipri > 2) print "Gentab=",gentab;
  if (ic == 1) alltab = gentab;
  else alltab = alltab |> gentab;
}
/* append zero column "level" */
w = cons(nrow(alltab),1);
alltab = alltab -> w;
cnam = [ "rowlev" "collev" "odds" "level" ];
alltab = cname(alltab,cnam);
if (ipri > 1) print "ALLTAB=",alltab;


modl = (gcrrow == 0 && gcrcol == 0) ? "3 = 4"
     : (gcrrow == 0) ? "3 = 2"
     : (gcrcol == 0) ? "3 = 1"
     : "3 = 1 2";
clas = [ 1 2 ];
optn = [ "print"          5 ,
         "link"       "log" ,
         "dist"       "nor" ,
         "desi"   "rankdef" ,
         "notype1"          ,
         "notype3"          ,
         "tech"     "trureg" ];
< gof,prm > = glim(alltab,modl,optn,clas);
if (ipri) {
  print "GOF=", gof;
  print "Parm=",prm;
}


if (ipri) print "nr0,nc0,ncm=",nr0,nc0,ncm;
free genprm; in = 0;
genprm[++in] = prm[1];
gennam = "gcr_delt";
ip = 2;
if (gcrrow) {
  for (ir = 1; ir < nrm; ir++, ip++) {
    genprm[++in] = prm[ip];
    gennam = gennam |> sprintf("row_gcr_%i",ir);
} }
if (gcrcol) {
  for (ic = 1; ic < ncm; ic++, ip++) {
```

```
    genprm[++in] = prm[ip];
    gennam = gennam |> sprintf("col_gcr_%i",ic);
} }

if (ngcrp) {
  genprm[++in] = gcr_oth = 0.;
  gennam = gennam |> "gcr_oth";
}
if (ipri) print "genprm=", genprm," gennam=", gennam;
np = nrow(genprm);
if (np > 1) {
  genprm = rname(genprm,gennam);
  if (ipri > 1) print "GENPRM=", genprm;
}

xini = parm |> genprm;
inam = rname(parm) |> gennam;
xini = rname(xini,inam);
if (ipri > 1) print "INIT=", xini;


/*--- Perform ML Optimization ---*/
zval = cons(nrc);
function loglik(x) global(zval,ipri,datc,
   rowpr2,colpr2,gcrpr2,cndpr2,gcrrow,gcrcol)
{
  /* parm x[ncm+nrm+nc0+1=9]:
     [npc=ncm=3] parms from col logistic
     [npr=nrm=2] parms from row logistic
     [ncm=3] parms from genmod
     for gcrp: [1] other gcr parm = 0 */
  /* datc[24,4] frequency table [ir,ic,i3,i4,i5]=[4,3,2] data:
               (pain=ir,med=ic,freq,ct,i3=,i4=,i5=vh) */
  nrc = nrow(datc);
  nr0 = datc[<>,1]; nc0 = datc[<>,2];
  nrm = nr0-1; ncm = nc0-1;
  nrowp = (rowpr2 == .) ? 0 : datc[<>,rowpr2];
  ncolp = (colpr2 == .) ? 0 : datc[<>,colpr2];
  ngcrp = (gcrpr2 == .) ? 0 : datc[<>,gcrpr2];
  ncond = (cndpr2 == .) ? 0 : datc[<>,cndpr2];
  npc = ncm; if (ncolp) ++npc;
  npr = nrm; if (nrowp) ++npr;

  /* start indices in parm vector x[]: */
  cind = 0; rind = cind + npc;
  igcr = ind = rind + npr + 1;
```

```
igc  = ind; if (gcrcol) ind += ncm;
igr  = ind; if (gcrrow) ind += nrm;
kgcr = ind;
if (ipri > 2) {
  print "npc,npr,colpr2,rowpr2=", npc,npr,colpr2,rowpr2;
  print "Indices=", cind,rind,igcr,igc,igr,kgcr;
}
/* 1. allocate: row_margin, col_margin, log_psi, erta, ksi */
eta = cons(nr0);
ksi = cons(nc0);
row_mrg = cons(nr0,1,0.);
col_mrg = cons(1,nc0,0.);
log_psi = cons(nr0,nc0,0.);
/* 2. allocate: */
cumpr = cons(nr0,nc0);
cellp = cons(nr0,nc0);
psi   = cons(nr0,nc0);
S_trm = cons(nr0,nc0);


llik = 0.;
for (ird = 1; ird <= nrc; ird++) {
  irow = datc[ird,1]; icol = datc[ird,2];
  for (i = 1; i < nr0; i++) {
    eta[i] = x[rind+i];
    if (nrowp) eta[i] += datc[ird,rowpr2] * x[rind+npr];
    row_mrg[i] = exp(eta[i]) / (1. + exp(eta[i]));
  }
  row_mrg[nr0] = 1;
  if (ipri > 2) print "Row_margin=",eta,row_mrg;
  for (j = 1; j < nc0; j++) {
    ksi[j] = x[cind+j];
    if (ncolp) ksi[j] += datc[ird,colpr2] * x[cind+npc];
    col_mrg[j] = exp(ksi[j]) / (1. + exp(ksi[j]));
  }
  col_mrg[nc0] = 1;
  if (ipri > 2) print "Col_margin=",ksi,col_mrg;
  for (i = 1; i <= nr0; i++) {
    tcon = x[igcr];                              /* GCRintercept */
    if (ngcrp) tcon += datc[ird,gcrpr2] * x[kgcr];     /* GCRpred */
    log_psi[i,ncm] = log_psi[i,nc0] = tcon;
    if (gcrrow) if (i < nr0) tcon += x[igr+i];        /* gcrlistr */
    for (j = 1; j < nc0; j++)
    log_psi[i,j] = (gcrcol) ? tcon + x[igc+j] : tcon; /* gcrlistc */
  }
  if (ipri > 2) print "Log_Psi=", log_psi;
```

```
cumpr[nr0,nc0] = 1.;
psi = exp(log_psi);
t1 = (1. + (row_mrg[1] + col_mrg[1])*(psi[1,1] - 1.))**2;
t2 = 4. * psi[1,1] * (1. - psi[1,1]) * row_mrg[1] * col_mrg[1];
S_trm[1,1] = sqrt(t1 + t2);
t1 = 1. + (row_mrg[1] + col_mrg[1])*(psi[1,1] - 1.) - S_trm[1,1];
t2 = 2. * (psi[1,1] - 1.);
cumpr[1,1] = t1 / t2;
cellp[1,1] = cumpr[1,1];
if (irow == 1 && icol == 1) goto lskip;


for (i = 2; i <= nr0; i++) {
  t1 = (1. + (row_mrg[i] + col_mrg[1]) * (psi[i,1] - 1.))**2;
  t2 = 4. * psi[i,1] * (1. - psi[i,1]) * row_mrg[i] * col_mrg[1];
  S_trm[i,1] = sqrt(t1 + t2);
  t1 = 1. + (row_mrg[i] + col_mrg[1]) * (psi[i,1] - 1.) - S_trm[i,1];
  t2 = 2. * (psi[i,1] - 1);
  cumpr[i,1] = t1 / t2;
  cellp[i,1] = cumpr[i,1] - cumpr[i-1,1];
}
if (icol == 1) goto lskip;

for (j = 2; j <= nc0; j++) {
  t1 = (1. + (row_mrg[1] + col_mrg[j]) * (psi[1,j] - 1.))**2;
  t2 = 4. * psi[1,j] * (1. - psi[1,j]) * row_mrg[1] * col_mrg[j];
  S_trm[1,j] = sqrt(t1 + t2);
  t1 = 1. + (row_mrg[1] + col_mrg[j]) * (psi[1,j] - 1.) - S_trm[1,j];
  t2 = 2. * (psi[1,j] - 1.);
  cumpr[1,j] = t1 / t2;
  cellp[1,j] = cumpr[1,j] - cumpr[1,j-1];
}
if (irow == 1) goto lskip;

for (i = 2; i <= nr0; i++)
for (j = 2; j <= nc0; j++) {
  t1 = (1. + (row_mrg[i] + col_mrg[j]) * (psi[i,j] - 1.))**2;
  t2 = 4. * psi[i,j] * (1. - psi[i,j]) * row_mrg[i] * col_mrg[j];
  S_trm[i,j] = sqrt(t1 + t2);
  t1 = 1. + (row_mrg[i] + col_mrg[j]) * (psi[i,j] - 1.) - S_trm[i,j];
  t2 = 2. * (psi[i,j] - 1.);
  cumpr[i,j] = t1 / t2;
  cellp[i,j] = cumpr[i,j]-cumpr[i-1,j]-cumpr[i,j-1]+cumpr[i-1,j-1];
}
```

```
lskip:
     /* Sum of Likelihoods: */
     zval[ird] = z = cellp[irow,icol];
     if (ipri > 2)
     print "ird,irow,icol,z,dat=", ird,irow,icol,z,datc[ird,4];
     llik += (z <= 1.e-8) ? 1.e100 : -log(z) * datc[ird,4];
   }
   if (ipri > 2) print "Loglike=", llik;
   return(llik);
 }


 /*---- Optimization --------------------------*/
 ipr = 1;
 ll1 = loglik(xini);
 if (ipri) print "Loglike Start=",ll1;

 /* This should be commented out */
 xres = [ 1.4349  1.9060  2.7289  1.0858  2.1677
          3.8333 -1.0806 -0.7872 -0.7617 ];
 ll2 = loglik(xres);
 if (ipri) print "Loglike Result=",ll2;

 /* Quasi-Newton */
 ipr = 0;
 mopt = [ "tech" "duquan" ];
 < xr, rp > = nlp(loglik,xini,mopt);
 if (ipri) print "Result parms=", xr;

 /* Newton-Ridge or Trust Region */
 ipr = 0;
 mopt = [ "tech" "trureg" ];
 < xr, rp > = nlp(loglik,xini,mopt);
 if (ipri) print "Result parms=", xr;


 /*--- Postprocessing ----------------------------*/
 /* Compute COV matrix of parm estimates and ASE */
 hess = fider(loglik,xr,"hess");
 hess = rname(hess,inam);
 hess = cname(hess,inam);
 if (ipri > 1) print "Hessian",hess;
 covp = inv(hess);
 covp = rname(covp,inam);
 covp = cname(covp,inam);
```

```
if (ipri) print "Covariance Matrix=",covp;
ase = sqrt(dia2vec(covp));
if (ipri > 1) print "ASE=", ase;

/* p value, t value, CIs */
ci = cons(nrow(xr),6);
cnam = [ "Parm" "ASE" "tval" "pval" "CIlow" "CIupp" ];
ci = cname(ci,cnam);
ci[,1] = xr‘;
ci[,2] = ase;
tval = xr‘ ./ ase;
ci[,3] = tval;  /* print "Tval=",tval; */
rdf = nrc; fval = tval .* tval;
pval = 1. - cdf("chis",fval,1); /* print "Pval=",pval; */
ci[,4] = pval;

alfa = .05; ta = 1. - .5 * alfa;
tt = pdf("norm",ta); /* print "Norm=",t; */
ci[,5] = xr‘ - tt * ase;
ci[,6] = xr‘ + tt * ase;
ci = cname(ci,cnam);
if (ipri) print "CI=",ci;


/*--- Compute Predicted Values and Residuals ---*/
ipr = 0;
llik = loglik(xr);
if (ipri) print "Loglike Result=",llik;
if (ipri) print "Zval=",zval;

yobs = datc[,4];
yprd = ares = sres = cons(nrc);
if (ncmp) {
  csum = cons(ncmp,1,0.);
  in = ii = 1;
  for (kk = 1; kk <= ncmp; kk++, in++) {
    sum = 0;
    for (ir = 1; ir <= nr0; ir++)
    for (ic = 1; ic <= nc0; ic++, ii++) sum += datc[ii,4];
    csum[in] = sum;
  }
  if (ipri > 1) print "CSUM=", csum;
  in = ii = 1;
  for (kk = 1; kk <= ncmp; kk++, in++) {
    for (ir = 1; ir <= nr0; ir++)
    for (ic = 1; ic <= nc0; ic++, ii++)
```

```
      yprd[ii] = csum[in] * zval[ii];
    }
  } else {
    sum = 0;
    for (ir = 1; ir <= nrc; ir++) sum += datc[ir,4];
    for (ir = 1; ir <= nrc; ir++) yprd[ir] = sum * zval[ir];
  }
  if (ipri) print "Ypred=",yprd;


  for (ir = 1; ir <= nrc; ir++) {
    t1 = yprd[ir];
    ares[ir] = t2 = yobs[ir] - t1;
    t3 = sqrt(t1 * (1. - zval[ir]));
    sres[ir] = t4 = (t3 == 0.) ? 0. : t2 / t3;
  }

  resm = datc[,1] -> datc[,2];
  for (j = 1; j <= nadc; j++) resm = resm -> datc[,4+j];
  resm = resm -> yobs -> yprd -> ares -> sres;
  cnam = colnam[1] -> colnam[2];
  for (j = 1; j <= nadc; j++) cnam = cnam -> colnam[4+j];
  cnam = cnam -> [ "yobs" "yprd" "resi" "stdres" ];
  resm = cname(resm,cnam);
  if (ipri) print "Residuals=",resm;

  /* Sort for last column */
  nc   = 6 + nadc;
  rank = resm[ >! ,nc];
  sres = resm[rank,];
  sres = cname(sres,cnam);
  if (ipri) print "Sorted Residuals=",sres;

  return(ci,resm);
}
```

Example 1 in the JSS paper is the example by Dale (1986) on page 913:

```
  /*-----------------------------------------------*/
  /*--- Example: Dale (1986), Table 3, page 913 ---*/
  /*-----------------------------------------------*/

  data = [ "VP"  1  170   7   8    0,
           "VP"  2   18   5   8    3,
           "VP"  3    7   0   4   14,
```

```
          "VA"  1  170   7   5    2,
          "VA"  2   22   7   8    1,
          "VA"  3    8   1   8    9,
          "VH"  1  176   8   5    2,
          "VH"  2   26   6   5    5,
          "VH"  3   14   3   2    9,
          "RA"  1  181   6   6    2,
          "RA"  2   17  12   7    3,
          "RA"  3   10   2   3   11 ];
  cnam = [ "Operation" "Pain" "Never" "Seldom" "Occ" "Reg" ] ;
  data = cname(data,cnam);
  print "Data Example Dale 1986=", data;


  /* Make string var "operation" to 4 binary vars */
  nr = nrow(data);
  uvp = data[,1] .== "VP"; print "VP=", uvp;
  uva = data[,1] .== "VA";
  uvh = data[,1] .== "VH";
  ura = data[,1] .== "RA";
  dat2 = uvp -> uva -> uvh -> ura -> data[,2:6];
  print "DAT2=", dat2;
  free uvp, uva, uvh, ura;
```

DAT2=

	1	2	3	4	5	6	7	8	9
1	1	0	0	0	1	170	7	8	0
2	1	0	0	0	2	18	5	8	3
3	1	0	0	0	3	7	0	4	14
4	0	1	0	0	1	170	7	5	2
5	0	1	0	0	2	22	7	8	1
6	0	1	0	0	3	8	1	8	9
7	0	0	1	0	1	176	8	5	2
8	0	0	1	0	2	26	6	5	5
9	0	0	1	0	3	14	3	2	9
10	0	0	0	1	1	181	6	6	2
11	0	0	0	1	2	17	12	7	3
12	0	0	0	1	3	10	2	3	11

```
  /*--- create (med,ct) vars from "Never" "Seldom" "Occ" "Reg" ---*/
  free datb;
  mapping = [ 1 2 3 4 0 1 ];
  rowmap = colmap = 0; gcrmap = 4; cndmap = .;
```

```
  row = cons(1,4);
  for (i = 1; i <= nr; i++) {
    row[1] = dat2[i,5];              /* rowvar=pain=[1] */
    row[4] = dat2[i,3]+1;            /* gcrpred=vh=[4] */
    row[2] = 1; row[3] = dat2[i,6];  /* colvar[2],count[3] */
    datb = datb |> row;
    row[2] = 2; row[3] = dat2[i,7];
    datb = datb |> row;
    row[2] = 3; row[3] = dat2[i,8];
    datb = datb |> row;
    row[2] = 4; row[3] = dat2[i,9];
    datb = datb |> row;
  }
  cnam = [ "Pain" "Med" "CT" "VH" ];
  datb = cname(datb,cnam);
  print "DATB=", datb;
```

```
DATB=

   |   Pain    Med     CT     VH
---------------------------------
 1 |      1      1    170      1
 2 |      1      2      7      1
 3 |      1      3      8      1
 4 |      1      4      0      1
 5 |      2      1     18      1
 6 |      2      2      5      1
 7 |      2      3      8      1
 8 |      2      4      3      1
 9 |      3      1      7      1
10 |      3      2      0      1
11 |      3      3      4      1
12 |      3      4     14      1
13 |      1      1    170      1
14 |      1      2      7      1
15 |      1      3      5      1
16 |      1      4      2      1
17 |      2      1     22      1
18 |      2      2      7      1
19 |      2      3      8      1
20 |      2      4      1      1
21 |      3      1      8      1
22 |      3      2      1      1
23 |      3      3      8      1
```

```
24 |      3      4      9      1
25 |      1      1    176      2
26 |      1      2      8      2
27 |      1      3      5      2
28 |      1      4      2      2
29 |      2      1     26      2
30 |      2      2      6      2
31 |      2      3      5      2
32 |      2      4      5      2
33 |      3      1     14      2
34 |      3      2      3      2
35 |      3      3      2      2
36 |      3      4      9      2
37 |      1      1    181      1
38 |      1      2      6      1
39 |      1      3      6      1
40 |      1      4      2      1
41 |      2      1     17      1
42 |      2      2     12      1
43 |      2      3      7      1
44 |      2      4      3      1
45 |      3      1     10      1
46 |      3      2      2      1
47 |      3      3      3      1
48 |      3      4     11      1
```

```
rowvar = 1; colvar = 2; count = 3;
/* 4 index vectors for var lists rowpred,... */
rowpred = colpred = cndpred = .; gcrpred = 4;
/* 2 binary vars: gcrrow,gcrcol */
gcrrow = 0; gcrcol = 1;
```

Te following call assumes that the BDM function is in the file `cmat\tana\bdm.inp`. Setting ipri to a positive integer will produce lots of printd output.

```
ipri = 2;
%inc "..\tana\bdm.inp";
lstvar(0); lstmem(0);
```

After data input, initialization and reading the macro we like to check our local variables and the memory usage until here:

```
Local Variables
***************
```

```
-------------------------------------------------------
        Name        Nest  Type           Content
-------------------------------------------------------
     1  data           1  MATRIX    STRING & INT
     2  cnam           1  VECTOR          STRING
     3  nr             1  INT                 12
     4  uvp            1  REAL           .
     5  uva            1  REAL           .
     6  uvh            1  REAL           .
     7  ura            1  REAL           .
     8  dat2           1  MATRIX             INT
     9  datb           0  MATRIX             INT
    10  mapping        1  VECTOR             INT
    11  colmap         1  INT                  0
    12  rowmap         1  INT                  0
    13  gcrmap         1  INT                  4
    14  cndmap         1  REAL           .
    15  row            1  VECTOR             INT
    16  i              1  INT                 13
    17  rowvar         1  INT                  1
    18  colvar         1  INT                  2
    19  count          1  INT                  3
    20  cndpred        1  REAL           .
    21  colpred        1  REAL           .
    22  rowpred        1  REAL           .
    23  gcrpred        1  INT                  4
    24  gcrrow         1  INT                  0
    25  gcrcol         1  INT                  1
    26  ipri           1  INT                  2
-------------------------------------------------------


   Summary Table of Memory Usage
   *****************************

 Memory Blocks Allocated:           5
 Number of Bytes Allocated:      7770
 Number of Bytes Used:           4232
 Number of Bytes Free:           3538

   List of 8 Largest Allocations
-------------------------------------------------------------
   Rank     Block     Begin    Length   Content
-------------------------------------------------------------
      1         3      3116      1692   data
```

```
      2         4      5231      1024   datb
      3         2      1001       688   dat2
      4         1         1       200   _COL_NAME
      5         3      4808       168   mapping
      6         1       201       164   cnam
      7         1       405       160   row
      8         4      6255       136   _COL_NAME
-------------------------------------------------------------
```

This is the call of the BDM macro:

```
< ci,resm > = BDM(ipri,datb,rowvar,colvar,count,
    rowpred,colpred,gcrpred,cndpred,gcrrow,gcrcol,
    rowmap,colmap,gcrmap,cndmap,mapping);
print "CI=", ci;
print "Residuals=", resm;
```

It follows a small part of the printed output:

1. the row logistic model:

```
                    **********************
                    Goodness of Model Fit
                    **********************

Log Likelihood     -749.14549      Degrees of Freedom         22
Deviance                .          Pearson ChiSquare          .
SSE                657.28757      MSE = SSE/nobs         27.386982

AIC (Intercept)    1502.2910      AIC (All Param.)       1502.2910
SBC (Intercept)    1512.1323      SBC (All Param.)       1512.1323
-2logL (Intercept) 1498.2910      -2logL (All Param.)    1498.2910
-2logL (ChiSqu.)   9.09e-013      Pvalue (df=      0)    0.0000000
Score ChiSqu. Test 1.27e-028      Pvalue (df=      0)    0.0000000

         Score Test for Proportional Odds Assumption:
ChiSquare          9.74e-029      Pvalue (df=      0)    0.0000000


         ****************************************
         Analysis of Effects and Parameter Estimates
         ****************************************

  Parameter        DF  Estimate Std_Error WaldChiSq Pr>ChiSq
```

```
Threshold      1  0  0.000000     .         .         .
Threshold      2  1  1.073758  0.072114 221.70229 0.000000
Threshold      3  1  2.157284  0.103078 438.00630 0.000000
Scale             0  1.000000  0.000000     .         .

    Covariance Matrix and ASE Based on Hessian Matrix
```

2. the column logistic model:

```
                    **********************
                    Goodness of Model Fit
                    **********************

Log Likelihood     -707.63420      Degrees of Freedom         21
Deviance                .          Pearson ChiSquare          .
SSE                 995.20804      MSE = SSE/nobs       41.467001

AIC (Intercept)     1421.2684      AIC (All Param.)     1421.2684
SBC (Intercept)     1436.0304      SBC (All Param.)     1436.0304
-2logL (Intercept)  1415.2684      -2logL (All Param.) 1415.2684
-2logL (ChiSqu.)    0.0000000      Pvalue (df=      0) 0.0000000
Score ChiSqu. Test  1.90e-028      Pvalue (df=      0) 0.0000000

         Score Test for Proportional Odds Assumption:
ChiSquare           2.02e-028      Pvalue (df=      0) 0.0000000


         ****************************************
         Analysis of Effects and Parameter Estimates
         ****************************************

  Parameter        DF  Estimate Std_Error WaldChiSq Pr>ChiSq

  Threshold      1  0  0.000000     .         .         .
  Threshold      2  1  1.440226  0.079848 325.33998 0.000000
  Threshold      3  1  1.915791  0.093940 415.90176 0.000000
  Threshold      4  1  2.747691  0.132075 432.80590 0.000000
  Scale             0  1.000000  0.000000     .         .

      Covariance Matrix and ASE Based on Hessian Matrix
```

3. input for another `glim` call (in SAS this is PROC GENMOD):

```
ALLTAB=
   |    rowlev    collev      odds     level
-------------------------------------------
 1 |    1.0000    1.0000   12.8289    0.0000
 2 |    1.0000    2.0000   13.9497    0.0000
 3 |    1.0000    3.0000   25.8983    0.0000
 4 |    2.0000    1.0000    9.8341    0.0000
 5 |    2.0000    2.0000   14.1664    0.0000
 6 |    2.0000    3.0000   27.1324    0.0000
```

4. some results of the `glim` (GENMOD) call:

```
                    **********************
                    Goodness of Model Fit
                    **********************

  Log Likelihood     -8.1239682      Degrees of Freedom          3
  Deviance            5.2691645      Scaled Deviance     6.0000000
  Pearson ChiSquare   5.2691645      Scaled Pearson CS   6.0000000
  SSE                 5.2691645      MSE = SSE/nobs      0.8781941
  AIC (All Param.)    22.247936      SBC (All Param.)    21.623215


          ****************************************
          Analysis of Effects and Parameter Estimates
          ****************************************

    Parameter        DF  Estimate Std_Error WaldChiSq Pr>ChiSq

    Intercept         1  3.277722  0.024991 17201.922 0.000000
    collev            1 -0.850135  0.063594 178.70643 0.000000
    collev            1 -0.634527  0.053351 141.45161 0.000000
    collev            0  0.000000     .         .        .
    Scale             1  0.937120  0.270523     .        .

        Covariance Matrix and ASE Based on Hessian Matrix


              ******************************
              Confidence Limits of Parameters
              ******************************

       Parameter          Estimate   LowWaldCL   UppWaldCL
```

```
Intercept         3.2777221  3.22874063  3.32670355
collev           -0.8501350 -0.97477737 -0.72549253
collev           -0.6345274 -0.73909437 -0.52996043
collev            0.0000000      .           .
Scale             0.9371201  0.40690423  1.46733597

 Wald Confidence Intervals Based on Hessian Matrix
```

5. Input into the optimization:

```
INIT=
              |         1
--------------------------
  col_Thresh1 |   1.44023
  col_Thresh2 |   1.91579
  col_Thresh3 |   2.74769
  row_Thresh1 |   1.07376
  row_Thresh2 |   2.15728
     gcr_delt |   3.27772
    col_gcr_1 |  -0.85013
    col_gcr_2 |  -0.63453
      gcr_oth |   0.00000
```

6. Results of the optimization:

   We implemented two calls of the `nlp` function for the same estimation: the Quasi_Newton method needs more iterations but is faster than the trust region method shown here:

```
                    Trust Region Optimization
                    Without Parameter Scaling
               Gradient Computed by Finite Differences
          Hessian Computed by Finite Differences (dense)
                    (Using Only Function Calls)


   Iteration Start:
   N. Variables              9
   Criterion       1314.771298        Max Grad Entry  6.456260408
   TR Radius       1.000000000


  Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius
     1    0    2   0  1312.179 2.592628 1.01040 1.43358  1.00000
```

```
  2    0    3   0  1311.457 0.721233 0.23921 0.00000  1.00109
  3    0    4   0  1311.457 4.2e-004  6e-004 0.00000  1.04338
  4    0    5   0  1311.457 1.8e-009  7e-005 0.00000   6e-003

Successful Termination After     4 Iterations
GCONV convergence criterion satisfied.
Criterion       1311.457017        Max Grad Entry  7.3510e-005
Ridge (lambda)  0.000000000        TR Radius       0.005941340
Act.dF/Pred.dF  1.010415060
N. Function Calls         6        N. Gradient Calls         2
N. Hessian Calls          6        Preproces. Time           2
Time for Method          10        Effective Time           14
```

7. The optimal parameter estimates:

```
                    ********************
                    Optimization Results
                    ********************

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

                Parameter     Estimate   Gradient

                 1 X1       1.43493283 -1.88e-005
                 2 X2       1.90603592  7.35e-005
                 3 X3       2.72889419 -1.23e-005
                 4 X4       1.08579780  1.46e-005
                 5 X5       2.16765535 -9.63e-006
                 6 X6       4.59503030  1.36e-005
                 7 X7      -1.84228692  0.0000000
                 8 X8      -1.54886378 -1.20e-005
                 9 X9      -0.76167932 -1.73e-005

            Value of Objective Function =      1311.46
```

8. The Hessian matrix is positive definite at the solution:

```
                        Hessian Matrix
                        **************

                    Symmetric Matrix: Dense Storage
```

S	1	2	3	4	5
1	459.76232	-293.70734	6.83e-004	-71.360381	1.7879550
2	-293.70734	438.78933	-100.29418	-25.377437	-15.322581
3	6.83e-004	-100.29418	139.26229	-4.6272380	-47.079209
4	-71.360381	-25.377437	-4.6272380	343.83478	-102.88736
5	1.7879550	-15.322581	-47.079209	-102.88736	178.05897
6	-21.196924	9.7225740	20.178217	-19.324284	2.8566861
7	-30.917118	52.599925	0.0000000	-16.809294	-3.1557017
8	9.7243327	-45.294401	30.429665	-0.6694833	-1.4008350
9	-7.9042627	4.8084972	-5.5210465	-3.8104088	8.8582760

S	6	7	8	9
1	-21.196924	-30.917118	9.7243327	-7.9042627
2	9.7225740	52.599925	-45.294401	4.8084972
3	20.178217	0.0000000	30.429665	-5.5210465
4	-19.324284	-16.809294	-0.6694833	-3.8104088
5	2.8566861	-3.1557017	-1.4008350	8.8582760
6	43.192452	30.104221	10.473658	13.850822
7	30.104221	78.011464	-47.910866	10.028355
8	10.473658	-47.910866	72.993517	-14.860241
9	13.850822	10.028355	-14.860241	31.370418

9. From the diagonal of the inverse Hessian matrix we obtain the asymptotic standard errors and the normal theory confidence intervals:

CI=

	Parm	ASE	tval	pval	CIlow	CIupp
1	1.4349	0.0795	18.0485	0.0000	1.2791	1.5908
2	1.9060	0.0932	20.4457	0.0000	1.7233	2.0888
3	2.7289	0.1303	20.9457	0.0000	2.4735	2.9842
4	1.0858	0.0713	15.2251	0.0000	0.9460	1.2256
5	2.1677	0.1014	21.3764	0.0000	1.9689	2.3664
6	4.5950	0.5332	8.6174	0.0000	3.5499	5.6401
7	-1.8423	0.4495	-4.0985	0.0000	-2.7233	-0.9613
8	-1.5489	0.4468	-3.4664	0.0005	-2.4246	-0.6731
9	-0.7617	0.3513	-2.1681	0.0302	-1.4502	-0.0731

10. The residuals are sorted w.r.t. decreasing values of absolute values of standardized residuals:

Sorted Residuals=

	Pain	Med	VH	yobs
1	3.000	2.000	0.000	3.000
2	2.000	2.000	0.000	24.000
3	1.000	3.000	1.000	5.000
4	1.000	3.000	0.000	19.000
5	2.000	1.000	0.000	57.000
6	3.000	1.000	1.000	14.000
7	3.000	2.000	1.000	3.000
8	3.000	3.000	1.000	2.000
9	2.000	2.000	1.000	6.000
10	2.000	4.000	1.000	5.000
11	2.000	3.000	1.000	5.000
12	1.000	2.000	1.000	8.000
13	3.000	1.000	0.000	25.000
14	1.000	2.000	0.000	20.000
15	2.000	1.000	1.000	26.000
16	2.000	4.000	0.000	7.000
17	3.000	3.000	0.000	15.000
18	3.000	4.000	1.000	9.000
19	1.000	4.000	0.000	4.000
20	1.000	4.000	1.000	2.000
21	1.000	1.000	1.000	176.000
22	3.000	4.000	0.000	34.000
23	2.000	3.000	0.000	23.000
24	1.000	1.000	0.000	521.000

	yprd	resi	stdres
1	6.480	-3.480	-1.373
2	18.766	5.234	1.224
3	8.033	-3.033	-1.087
4	14.904	4.096	1.072
5	63.025	-6.025	-0.793
6	11.700	2.300	0.688
7	2.028	0.972	0.685
8	3.222	-1.222	-0.685
9	4.659	1.341	0.627
10	3.807	1.193	0.616
11	6.529	-1.529	-0.606
12	9.732	-1.732	-0.566
13	22.615	2.385	0.509
14	22.059	-2.059	-0.445
15	24.082	1.918	0.410

16	8.091	-1.091	-0.386
17	13.629	1.371	0.375
18	9.853	-0.853	-0.277
19	3.497	0.503	0.269
20	2.337	-0.337	-0.221
21	175.019	0.981	0.129
22	34.502	-0.502	-0.087
23	22.706	0.294	0.063
24	521.727	-0.727	-0.058

11. The following are the returned arguments of the `bdm` function:

CI=

	Parm	ASE	tval	pval	CIlow	CIupp
1	1.4349	0.0795	18.0485	0.0000	1.2791	1.5908
2	1.9060	0.0932	20.4457	0.0000	1.7233	2.0888
3	2.7289	0.1303	20.9457	0.0000	2.4735	2.9842
4	1.0858	0.0713	15.2251	0.0000	0.9460	1.2256
5	2.1677	0.1014	21.3764	0.0000	1.9689	2.3664
6	4.5950	0.5332	8.6174	0.0000	3.5499	5.6401
7	-1.8423	0.4495	-4.0985	0.0000	-2.7233	-0.9613
8	-1.5489	0.4468	-3.4664	0.0005	-2.4246	-0.6731
9	-0.7617	0.3513	-2.1681	0.0302	-1.4502	-0.0731

Residuals=

	Pain	Med	VH	yobs
1	1.000	1.000	0.000	521.000
2	1.000	2.000	0.000	20.000
3	1.000	3.000	0.000	19.000
4	1.000	4.000	0.000	4.000
5	2.000	1.000	0.000	57.000
6	2.000	2.000	0.000	24.000
7	2.000	3.000	0.000	23.000
8	2.000	4.000	0.000	7.000
9	3.000	1.000	0.000	25.000
10	3.000	2.000	0.000	3.000
11	3.000	3.000	0.000	15.000
12	3.000	4.000	0.000	34.000
13	1.000	1.000	1.000	176.000
14	1.000	2.000	1.000	8.000
15	1.000	3.000	1.000	5.000

```
16 |     1.000     4.000     1.000     2.000
17 |     2.000     1.000     1.000    26.000
18 |     2.000     2.000     1.000     6.000
19 |     2.000     3.000     1.000     5.000
20 |     2.000     4.000     1.000     5.000
21 |     3.000     1.000     1.000    14.000
22 |     3.000     2.000     1.000     3.000
23 |     3.000     3.000     1.000     2.000
24 |     3.000     4.000     1.000     9.000
```

```
   |      yprd      resi    stdres
-----------------------------------
 1 |   521.727    -0.727    -0.058
 2 |    22.059    -2.059    -0.445
 3 |    14.904     4.096     1.072
 4 |     3.497     0.503     0.269
 5 |    63.025    -6.025    -0.793
 6 |    18.766     5.234     1.224
 7 |    22.706     0.294     0.063
 8 |     8.091    -1.091    -0.386
 9 |    22.615     2.385     0.509
10 |     6.480    -3.480    -1.373
11 |    13.629     1.371     0.375
12 |    34.502    -0.502    -0.087
13 |   175.019     0.981     0.129
14 |     9.732    -1.732    -0.566
15 |     8.033    -3.033    -1.087
16 |     2.337    -0.337    -0.221
17 |    24.082     1.918     0.410
18 |     4.659     1.341     0.627
19 |     6.529    -1.529    -0.606
20 |     3.807     1.193     0.616
21 |    11.700     2.300     0.688
22 |     2.028     0.972     0.685
23 |     3.222    -1.222    -0.685
24 |     9.853    -0.853    -0.277
```

The second example in the JSS paper is the example by Dale 1985, Table 2: model a, page 32:

```
dale1985 = [  1  0  1 0  0 0  0,      1  1  0  0  0  0  0,
              1  2  0 0  0 0  0,      2  0  2  0  0  0  0,
              2  1  0 0  0 0  0,      2  2  0  0  0  0  0,
              3  0  5 0  0 0  0,      3  1  0  0  0  0  0,
```

```
 3  2  0 0  0 0  0,      4  0  9  2  1  0  0,
 4  1  3 1  0 0  0,      4  2  0  0  0  0  0,
 5  0  7 2  0 0  0,      5  1  0  0  0  1  0,
 5  2  0 0  0 0  0,      6  0  4  0  0  0  0,
 6  1  1 2  0 0  0,      6  2  0  0  0  0  0,
 7  0  2 0  0 0  0,      7  1  3  0  0  0  0,
 7  2  0 0  0 0  0,      8  0  6  1  0  0  0,
 8  1  2 2  2 2  0,      8  2  0  0  0  0  0,
10  0  0 0  0 0  0,     10  1  0  2  0  1  0,
10  2  0 0  0 0  0,     11  0  0  0  0  0  0,
11  1  3 0  0 0  0,     11  2  0  0  1  0  0,
12  0  0 0  0 0  0,     12  1  1  2  0  0  0,
12  2  0 0  0 1  0,     14  0  1  0  0  0  0,
14  1  1 2  1 2  0,     14  2  0  0  1  2  4,
15  0  1 0  0 0  0,     15  1  1  1  0  0  0,
15  2  0 0  0 0  0,     17  0  0  0  0  0  0,
17  1  0 0  0 1  0,     17  2  0  0  0  0  0,
18  0  0 0  0 0  0,     18  1  0  0  0  1  0,
18  2  0 0  0 0  0 ];
```

```
 cnam = [ "Period" "Orbit" "MAN0" "MAN1" "MAN2" "MAN3" "MAN4" ];
 dale = cname(dale1985,cnam);
 print "Data Example Dale 1985=", dale;
```

```
/*--- (row=orbit,col=mand,count=ct, rowpred=colpred=period)
      from "MAN0" "MAN1" "MAN2" "MAN3" "MAN4" ---*/
free datb;
irowmap = [ 0 0 0 1 2   3 0 4 0 0
            5 6 0 7 0   0 0 0 ];
mapping = [ 4  5  6  8 11 12 14 ];
rowmap = colmap = 0; gcrmap = cndmap = .;
```

```
nr = nrow(dale); row = cons(1,4);
for (i = 1; i <= nr; i++) {
  dp = irowmap[dale[i,1]]; if (dp == 0) continue;
  row[1] = dale[i,2]+1;     /* rowvar[1]=orbit[2] */
  row[2] = 1;               /* colvar[2]          */
  row[3] = dale[i,3];       /* count[3]=ct=MAN0   */
  row[4] = dp;              /* rowpred[4]=colpred=dp */
  datb = datb |> row;
  row[2] = 2;                    /* colvar[2] */
  row[3] = dale[i,4]+dale[i,5];  /* count[3]=ct=MAN1+MAN2 */
  datb = datb |> row;
```

```
  row[2] = 3;                    /* colvar[2] */
  row[3] = dale[i,6]+dale[i,7];  /* count[3]=ct=MAN3+MAN4 */
  datb = datb |> row;
}
cnam = [ "Orbit" "Mand" "CT" "Period" ];
datb = cname(datb,cnam);
print "DATB=", datb;


rowvar = 1; colvar = 2; count = 3;
/* 4 index vectors for var lists rowpred,... */
rowpred = colpred = 4; gcrpred = cndpred = .;
/* 2 binary vars: gcrrow,gcrcol */
gcrrow = gcrcol = 0;
```

The `bdm` call is the same as above in example 1:

```
ipri = 2;
%inc "..\tana\bdm.inp";
lstvar(0);
lstmem(0);

 < ci,resm > = BDM(ipri,datb,rowvar,colvar,count,
     rowpred,colpred,gcrpred,cndpred,gcrrow,gcrcol,
     rowmap,colmap,gcrmap,cndmap,mapping);
 print "CI=", ci;
 print "Residuals=", resm;
```

It follows a small part of the printed output:

1. The output of the row logistic model:

```
                      **********
                      Goodness of Model Fit
                      **********

     Log Likelihood     -47.902538      Degrees of Freedom         23
     Deviance              .            Pearson ChiSquare       .
     SSE                54.108709      MSE = SSE/nobs       0.8588684

     AIC (Intercept)    141.86634      AIC (All Param.)     101.80508
     SBC (Intercept)    146.36333      SBC (All Param.)     108.55056
     -2logL (Intercept) 137.86634      -2logL (All Param.) 95.805076
     -2logL (ChiSqu.)   42.061264      Pvalue (df=      1) 0.0000000
     Score ChiSqu. Test 37.666476      Pvalue (df=      1) 0.0000000
```

```
        Score Test for Proportional Odds Assumption:
ChiSquare          0.9933388      Pvalue (df=      1) 0.3189277


        ****************************************
        Analysis of Effects and Parameter Estimates
        ****************************************

  Parameter        DF  Estimate Std_Error WaldChiSq Pr>ChiSq

  Threshold      1  0  0.000000     .        .         .
  Threshold      2  1  3.781721  0.791451 22.831308 0.000002
  Threshold      3  1  7.449133  1.339199 30.940079 0.000000
  Period            1 -0.527037  0.105075 25.158504 0.000001
  Scale             0  1.000000  0.000000     .        .

      Covariance Matrix and ASE Based on Hessian Matrix


             ******************************
             Confidence Limits of Parameters
             ******************************

    Parameter          Estimate   LowWaldCL   UppWaldCL

    Intercept      1  3.7817213  2.23050551  5.33293702
    Intercept      2  7.4491328  4.82435163  10.0739139
    Period           -0.5270370 -0.73297983 -0.32109410

     Wald Confidence Intervals Based on Hessian Matrix
```

2. The output of the column logistic model:

```
                    ********************
                    Goodness of Model Fit
                    ********************

Log Likelihood     -59.662148      Degrees of Freedom         23
Deviance              .            Pearson ChiSquare       .
SSE                63.986981      MSE = SSE/nobs       1.0156664

AIC (Intercept)    142.86574      AIC (All Param.)     125.32430
SBC (Intercept)    147.36273      SBC (All Param.)     132.06978
```

```
-2logL (Intercept)  138.86574      -2logL (All Param.) 119.32430
-2logL (ChiSqu.)    19.541440      Pvalue (df=      1) 0.0000098
Score ChiSqu. Test  19.503969      Pvalue (df=      1) 0.0000100

         Score Test for Proportional Odds Assumption:
ChiSquare           1.2275590      Pvalue (df=      1) 0.2678823


         ****************************************
         Analysis of Effects and Parameter Estimates
         ****************************************

  Parameter        DF  Estimate Std_Error WaldChiSq Pr>ChiSq

  Threshold      1  0  0.000000     .         .          .
  Threshold      2  1  2.507714  0.634473 15.621775 0.000077
  Threshold      3  1  4.308789  0.796128 29.291679 0.000000
  Period            1 -0.295318  0.071540 17.040680 0.000037
  Scale             0  1.000000  0.000000     .         .

      Covariance Matrix and ASE Based on Hessian Matrix
```

3. The `glim` call substituting the PROC GENMOD call:

```
ALLTAB=
   |    rowlev    collev      odds     level
-------------------------------------------
 1 |   1.00000   1.00000   3.67200   0.00000
 2 |   1.00000   2.00000   3.29380   0.00000
 3 |   2.00000   1.00000   3.35014   0.00000
 4 |   2.00000   2.00000   3.86842   0.00000


                     **********************
                     Goodness of Model Fit
                     **********************

  Log Likelihood      0.1088202      Degrees of Freedom          3
  Deviance            0.2217970      Scaled Deviance     4.0000001
  Pearson ChiSquare   0.2217970      Scaled Pearson CS   4.0000001
  SSE                 0.2217970      MSE = SSE/nobs      0.0554492
  AIC (All Param.)    1.7823597      SBC (All Param.)    1.1686541
```

```
        ****************************************
        Analysis of Effects and Parameter Estimates
        ****************************************

Parameter        DF  Estimate Std_Error WaldChiSq Pr>ChiSq

Intercept         1  1.265846  0.033202 1453.5350 0.000000
level             0  0.000000     .         .         .
Scale             1  0.235477  0.083254     .         .

    Covariance Matrix and ASE Based on Hessian Matrix


            ******************************
            Confidence Limits of Parameters
            ******************************

   Parameter          Estimate   LowWaldCL   UppWaldCL

   Intercept         1.2658457  1.20077041  1.33092099
   level             0.0000000      .           .
   Scale             0.2354766  0.07230265  0.39865062

    Wald Confidence Intervals Based on Hessian Matrix
```

4. Start values for the optimization:

```
INIT=
              |         1
--------------------------
  col_Thresh1 |   2.50771
  col_Thresh2 |   4.30879
   col_Period |  -0.29532
  row_Thresh1 |   3.78172
  row_Thresh2 |   7.44913
   row_Period |  -0.52704
     gcr_delt |   1.26585
```

5. History output of the optimization process:

```
                Trust Region Optimization
                Without Parameter Scaling
          Gradient Computed by Finite Differences
```

```
      Hessian Computed by Finite Differences (dense)
                (Using Only Function Calls)


 Iteration Start:
 N. Variables              7
 Criterion       100.7102033        Max Grad Entry  3.212619347
 TR Radius       1.000000000

Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius
   1    0    2   0  99.14077 1.569430 0.64666 0.00000  1.00000
   2    0    3   0  99.13503 5.7e-003  6e-003 0.00000  0.94718
   3    0    4   0  99.13503 7.4e-007  7e-006 0.00000  0.07854

 Successful Termination After     3 Iterations
 GCONV convergence criterion satisfied.
 Criterion       99.13502871        Max Grad Entry  6.6637e-006
 Ridge (lambda)  0.000000000        TR Radius       0.078544006
 Act.dF/Pred.dF  0.999671935
 N. Function Calls         5        N. Gradient Calls         2
 N. Hessian Calls          5        Preproces. Time           5
 Time for Method          16        Effective Time           26
```

6. The optimal parmeter estimates:

```
                    ********************
                    Optimization Results
                    ********************

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

                Parameter     Estimate   Gradient

                  1 X1       2.43509105  2.78e-007
                  2 X2       4.25603868 -3.63e-007
                  3 X3      -0.28804277  6.66e-006
                  4 X4       3.75160976  8.03e-007
                  5 X5       7.47879715  4.50e-007
                  6 X6      -0.52252075  2.51e-006
                  7 X7       2.25043639  8.80e-007

          Value of Objective Function =       99.135
```

7. The Hessian matrix is positive definite:

Hessian Matrix
****

Symmetric Matrix: Dense Storage

S	1	2	3	4	5
1	19.261795	-4.0843080	110.28760	-6.1705503	-0.5938027
2	-4.0843080	11.070423	84.379993	-0.5937073	-2.9332580
3	110.28760	84.379993	2024.6281	-45.693201	-45.967866
4	-6.1705503	-0.5937073	-45.693201	13.892949	-0.4552035
5	-0.5938027	-2.9332580	-45.967866	-0.4552035	6.6562915
6	-49.367776	-42.294638	-961.47010	90.004842	81.258067
7	-0.3159594	0.5356457	1.5054122	-0.2530330	-0.0914882

S	6	7
1	-49.367776	-0.3159594
2	-42.294638	0.5356457
3	-961.47010	1.5054122
4	90.004842	-0.2530330
5	81.258067	-0.0914882
6	1764.3137	-3.6038644
7	-3.6038644	3.0590642

The residual sorted w.r.t. decreasing values of standardized residuals:

Sorted Residuals=

	Orbit	Mand	Period	yobs
1	2.0000	1.0000	11.0000	3.0000
2	3.0000	2.0000	11.0000	1.0000
3	2.0000	1.0000	4.0000	3.0000
4	1.0000	1.0000	14.0000	1.0000
5	2.0000	2.0000	11.0000	0.0000
6	1.0000	1.0000	4.0000	9.0000
7	1.0000	2.0000	4.0000	3.0000
8	2.0000	2.0000	5.0000	0.0000
9	1.0000	2.0000	5.0000	2.0000
10	2.0000	3.0000	5.0000	1.0000
11	2.0000	1.0000	5.0000	0.0000

12	2.0000	2.0000	6.0000	2.0000
13	1.0000	2.0000	6.0000	0.0000
14	2.0000	3.0000	12.0000	0.0000
15	2.0000	3.0000	11.0000	0.0000
16	3.0000	3.0000	14.0000	6.0000
17	2.0000	1.0000	14.0000	1.0000
18	3.0000	3.0000	11.0000	0.0000
19	2.0000	1.0000	8.0000	2.0000
20	1.0000	1.0000	11.0000	0.0000

21	2.0000	3.0000	6.0000	0.0000
22	2.0000	3.0000	4.0000	0.0000
23	2.0000	2.0000	12.0000	2.0000
24	1.0000	1.0000	8.0000	6.0000
25	3.0000	3.0000	8.0000	0.0000
26	1.0000	3.0000	4.0000	0.0000
27	3.0000	1.0000	14.0000	0.0000
28	3.0000	2.0000	12.0000	0.0000
29	2.0000	3.0000	8.0000	2.0000
30	3.0000	3.0000	12.0000	1.0000
31	1.0000	1.0000	12.0000	0.0000
32	3.0000	2.0000	14.0000	1.0000
33	2.0000	2.0000	14.0000	3.0000
34	3.0000	2.0000	8.0000	0.0000
35	1.0000	3.0000	5.0000	0.0000
36	1.0000	1.0000	5.0000	7.0000
37	1.0000	3.0000	8.0000	0.0000
38	2.0000	3.0000	14.0000	2.0000
39	1.0000	2.0000	14.0000	0.0000
40	1.0000	3.0000	6.0000	0.0000

41	1.0000	2.0000	11.0000	0.0000
42	2.0000	1.0000	12.0000	1.0000
43	1.0000	2.0000	12.0000	0.0000
44	3.0000	1.0000	8.0000	0.0000
45	3.0000	1.0000	12.0000	0.0000
46	3.0000	3.0000	6.0000	0.0000
47	3.0000	2.0000	6.0000	0.0000
48	3.0000	1.0000	11.0000	0.0000
49	3.0000	2.0000	5.0000	0.0000
50	3.0000	2.0000	4.0000	0.0000
51	1.0000	3.0000	14.0000	0.0000
52	2.0000	1.0000	6.0000	1.0000
53	3.0000	3.0000	5.0000	0.0000

54	1.0000	2.0000	8.0000	1.0000
55	3.0000	3.0000	4.0000	0.0000
56	3.0000	1.0000	4.0000	0.0000
57	1.0000	1.0000	6.0000	4.0000
58	1.0000	3.0000	11.0000	0.0000
59	3.0000	1.0000	5.0000	0.0000
60	3.0000	1.0000	6.0000	0.0000
61	1.0000	3.0000	12.0000	0.0000
62	2.0000	2.0000	4.0000	1.0000
63	2.0000	2.0000	8.0000	4.0000

	yprd	resi	stdres
1	0.8929	2.1071	2.5301
2	0.1620	0.8380	2.1257
3	0.9718	2.0282	2.1228
4	0.2451	0.7549	1.5384
5	1.4420	-1.4420	-1.5016
6	11.5351	-2.5351	-1.4130
7	1.6414	1.3586	1.1195
8	1.0955	-1.0955	-1.1092
9	1.0069	0.9931	1.0436
10	0.3802	0.6198	1.0249
11	0.8729	-0.8729	-0.9779
12	1.1254	0.8746	0.8999
13	0.6328	-0.6328	-0.8341
14	0.5850	-0.5850	-0.8278
15	0.5848	-0.5848	-0.8276
16	4.6180	1.3820	0.7856
17	1.8607	-0.8607	-0.6776
18	0.4045	-0.4045	-0.6709
19	3.0198	-1.0198	-0.6567
20	0.3696	-0.3696	-0.6382
21	0.3834	-0.3834	-0.6368
22	0.3952	-0.3952	-0.6366
23	1.4025	0.5975	0.6260
24	4.9092	1.0908	0.6002
25	0.2872	-0.2872	-0.5411
26	0.2706	-0.2706	-0.5247
27	0.2508	-0.2508	-0.5053
28	0.2339	-0.2339	-0.4985
29	1.4359	0.5641	0.4951

```
30 |    0.6407    0.3593    0.4898
31 |    0.2194   -0.2194   -0.4818
32 |    1.5602   -0.5602   -0.4758
33 |    3.7525   -0.7525   -0.4540
34 |    0.1865   -0.1865   -0.4346
35 |    0.1576   -0.1576   -0.4002
36 |    6.4104    0.5896    0.3887
37 |    0.1424   -0.1424   -0.3792
38 |    1.5721    0.4279    0.3622
39 |    0.1095   -0.1095   -0.3322
40 |    0.0969   -0.0969   -0.3135


41 |    0.0899   -0.0899   -0.3032
42 |    0.7944    0.2056    0.2577
43 |    0.0640   -0.0640   -0.2551
44 |    0.0605   -0.0605   -0.2465
45 |    0.0452   -0.0452   -0.2138
46 |    0.0372   -0.0372   -0.1935
47 |    0.0363   -0.0363   -0.1911
48 |    0.0353   -0.0353   -0.1887
49 |    0.0323   -0.0323   -0.1800
50 |    0.0312   -0.0312   -0.1768
51 |    0.0311   -0.0311   -0.1767
52 |    0.8556    0.1444    0.1667
53 |    0.0269   -0.0269   -0.1643
54 |    0.8654    0.1346    0.1490
55 |    0.0213   -0.0213   -0.1461
56 |    0.0202   -0.0202   -0.1423
57 |    3.8162    0.1838    0.1395
58 |    0.0189   -0.0189   -0.1379
59 |    0.0172   -0.0172   -0.1313
60 |    0.0162   -0.0162   -0.1273
61 |    0.0148   -0.0148   -0.1218
62 |    1.1131   -0.1131   -0.1111
63 |    4.0929   -0.0929   -0.0539


CI=
   |      Parm       ASE      tval      pval     CIlow     CIupp
-------------------------------------------------------------------
 1 |    2.4351    0.6282    3.8762    0.0001    1.2038    3.6664
 2 |    4.2560    0.7978    5.3349    1e-007    2.6924    5.8196
 3 |   -0.2880    0.0707   -4.0734    0.0000   -0.4266   -0.1494
 4 |    3.7516    0.7736    4.8495    1e-006    2.2354    5.2678
 5 |    7.4788    1.3083    5.7164    1e-008    4.9146   10.0430
```

6	-0.5225	0.1010	-5.1751	2e-007	-0.7204	-0.3246
7	2.2504	0.5766	3.9028	0.0001	1.1203	3.3806

Residuals=

	Orbit	Mand	Period	yobs
1	1.0000	1.0000	4.0000	9.0000
2	1.0000	2.0000	4.0000	3.0000
3	1.0000	3.0000	4.0000	0.0000
4	2.0000	1.0000	4.0000	3.0000
5	2.0000	2.0000	4.0000	1.0000
6	2.0000	3.0000	4.0000	0.0000
7	3.0000	1.0000	4.0000	0.0000
8	3.0000	2.0000	4.0000	0.0000
9	3.0000	3.0000	4.0000	0.0000
10	1.0000	1.0000	5.0000	7.0000
11	1.0000	2.0000	5.0000	2.0000
12	1.0000	3.0000	5.0000	0.0000
13	2.0000	1.0000	5.0000	0.0000
14	2.0000	2.0000	5.0000	0.0000
15	2.0000	3.0000	5.0000	1.0000
16	3.0000	1.0000	5.0000	0.0000
17	3.0000	2.0000	5.0000	0.0000
18	3.0000	3.0000	5.0000	0.0000
19	1.0000	1.0000	6.0000	4.0000
20	1.0000	2.0000	6.0000	0.0000
21	1.0000	3.0000	6.0000	0.0000
22	2.0000	1.0000	6.0000	1.0000
23	2.0000	2.0000	6.0000	2.0000
24	2.0000	3.0000	6.0000	0.0000
25	3.0000	1.0000	6.0000	0.0000
26	3.0000	2.0000	6.0000	0.0000
27	3.0000	3.0000	6.0000	0.0000
28	1.0000	1.0000	8.0000	6.0000
29	1.0000	2.0000	8.0000	1.0000
30	1.0000	3.0000	8.0000	0.0000
31	2.0000	1.0000	8.0000	2.0000
32	2.0000	2.0000	8.0000	4.0000
33	2.0000	3.0000	8.0000	2.0000
34	3.0000	1.0000	8.0000	0.0000
35	3.0000	2.0000	8.0000	0.0000
36	3.0000	3.0000	8.0000	0.0000
37	1.0000	1.0000	11.0000	0.0000

38	1.0000	2.0000	11.0000	0.0000
39	1.0000	3.0000	11.0000	0.0000
40	2.0000	1.0000	11.0000	3.0000
41	2.0000	2.0000	11.0000	0.0000
42	2.0000	3.0000	11.0000	0.0000
43	3.0000	1.0000	11.0000	0.0000
44	3.0000	2.0000	11.0000	1.0000
45	3.0000	3.0000	11.0000	0.0000
46	1.0000	1.0000	12.0000	0.0000
47	1.0000	2.0000	12.0000	0.0000
48	1.0000	3.0000	12.0000	0.0000
49	2.0000	1.0000	12.0000	1.0000
50	2.0000	2.0000	12.0000	2.0000
51	2.0000	3.0000	12.0000	0.0000
52	3.0000	1.0000	12.0000	0.0000
53	3.0000	2.0000	12.0000	0.0000
54	3.0000	3.0000	12.0000	1.0000
55	1.0000	1.0000	14.0000	1.0000
56	1.0000	2.0000	14.0000	0.0000
57	1.0000	3.0000	14.0000	0.0000
58	2.0000	1.0000	14.0000	1.0000
59	2.0000	2.0000	14.0000	3.0000
60	2.0000	3.0000	14.0000	2.0000
61	3.0000	1.0000	14.0000	0.0000
62	3.0000	2.0000	14.0000	1.0000
63	3.0000	3.0000	14.0000	6.0000

	yprd	resi	stdres
1	11.5351	-2.5351	-1.4130
2	1.6414	1.3586	1.1195
3	0.2706	-0.2706	-0.5247
4	0.9718	2.0282	2.1228
5	1.1131	-0.1131	-0.1111
6	0.3952	-0.3952	-0.6366
7	0.0202	-0.0202	-0.1423
8	0.0312	-0.0312	-0.1768
9	0.0213	-0.0213	-0.1461
10	6.4104	0.5896	0.3887
11	1.0069	0.9931	1.0436
12	0.1576	-0.1576	-0.4002
13	0.8729	-0.8729	-0.9779
14	1.0955	-1.0955	-1.1092

15	0.3802	0.6198	1.0249
16	0.0172	-0.0172	-0.1313
17	0.0323	-0.0323	-0.1800
18	0.0269	-0.0269	-0.1643
19	3.8162	0.1838	0.1395
20	0.6328	-0.6328	-0.8341
21	0.0969	-0.0969	-0.3135
22	0.8556	0.1444	0.1667
23	1.1254	0.8746	0.8999
24	0.3834	-0.3834	-0.6368
25	0.0162	-0.0162	-0.1273
26	0.0363	-0.0363	-0.1911
27	0.0372	-0.0372	-0.1935
28	4.9092	1.0908	0.6002
29	0.8654	0.1346	0.1490
30	0.1424	-0.1424	-0.3792
31	3.0198	-1.0198	-0.6567
32	4.0929	-0.0929	-0.0539
33	1.4359	0.5641	0.4951
34	0.0605	-0.0605	-0.2465
35	0.1865	-0.1865	-0.4346
36	0.2872	-0.2872	-0.5411
37	0.3696	-0.3696	-0.6382
38	0.0899	-0.0899	-0.3032
39	0.0189	-0.0189	-0.1379
40	0.8929	2.1071	2.5301
41	1.4420	-1.4420	-1.5016
42	0.5848	-0.5848	-0.8276
43	0.0353	-0.0353	-0.1887
44	0.1620	0.8380	2.1257
45	0.4045	-0.4045	-0.6709
46	0.2194	-0.2194	-0.4818
47	0.0640	-0.0640	-0.2551
48	0.0148	-0.0148	-0.1218
49	0.7944	0.2056	0.2577
50	1.4025	0.5975	0.6260
51	0.5850	-0.5850	-0.8278
52	0.0452	-0.0452	-0.2138
53	0.2339	-0.2339	-0.4985
54	0.6407	0.3593	0.4898
55	0.2451	0.7549	1.5384
56	0.1095	-0.1095	-0.3322

57	0.0311	-0.0311	-0.1767
58	1.8607	-0.8607	-0.6776
59	3.7525	-0.7525	-0.4540
60	1.5721	0.4279	0.3622
61	0.2508	-0.2508	-0.5053
62	1.5602	-0.5602	-0.4758
63	4.6180	1.3820	0.7856