CMAT Newsletter: July 2003

Wolfgang M. Hartmann

July 19, 2007

1 General Remarks

Now, a **CMAT User Forum** is available on our website. For that, I did need a CMAT logo which can be seen there. This is easy with Acrobat, took me less than five minutes. Questions for installation and running CMAT can be posted there. I will try to respond whenever it is necessary. However, all users are encouraged to respond with their own experience. Hopefully, this Forum will improve CMAT and attract a larger user community. Maybe some of you could do me the favour and post something in the forum just to see if it works. I get the forum hosting for free as long as I accept the small advertisement there and a minimum of people is checking up on it. (Otherwise I would have to pay some small fees.) Please, write something nice that I know somebody likes what I'm doing, even if you don't use CMAT at this time.

1.1 Fixed Bugs

A number of bugs were fixed:

1. Memory crash in `univar` when `"min"` or `"max"` was requested.
2. Indexing of matrices with string and numeric entries.
3. The `nomiss` option for the `svm` function.
4. Kronecker product `c = a @ b;` of two symmetric matrices `a` and `b`.
5. Indexing subarray of diagonal matrix `d`, i.e. obtaining `dsub = d[1:5,1:5];` where `d` is diagonal matrix.
6. The `(int)` and `(real)` casts did now work properly for string data. However, it was always possible to use the `sscanf` function for changing strings to numeric values. See example in the *Illustration* section below.

1.2 Changes in `univar()`

The result now has row and column names. Therefore, I was able to drop the string content in the first column. The computation of the median and other quantiles is now much faster for objects with many columns (variables).

1.3 New Features: `sem()`

The `sem` function was extended by the `rob` option. Specifying the `rob` option is valid only for methods ULS, ML, and GLS and raw input data which are not multivariate normal distributed. The following additional statistics is computed

- robust approx. standard errors (Satorra-Bentler corrected)
- mean corrected robust χ^2 value

- mean-variance corrected robust χ^2 value

which is based on data corrected for deviations from normality. The option is not valid for input correlation or covariance matrices. Older features for treating nonnormal data where the WLS and DWLS estimation methods and the Bollen-Stine bootstrap method.
Comparison with LISREL 8.54 and Mplus 2.13:

LISREL: ML and GLS our robust ASEs are very close to those of LISREL (about 4 decimals). The Satorra-Bentler mean corrected χ^2 is moderate close to that of LISREL.

Mplus: ML our results differ slightly from Mplus. But there are already differences in the parameter estimates starting in the second decimal. Since derivatives are sensitive to the precision of the parameter estimates the differences for ASE's and the S-B mean corrected χ^2 are not surprising.

LISREL: ULS our robust ASEs are in general smaller than those of LISREL. On the other hand our S-B mean corrected χ^2 is larger. There seems to be a problem either in our code or in that of LISREL. (There is no robust ULS in Mplus.)

Unfortunately we are not able to compare our results to those of EQS since we don't have a recent copy of EQS.

1.4 New Features: `svm()`

Another input argument `ctun` can be specified which increases the number of input arguments to eight:

alfa = svm(train,model,<,optn<,class<,ctun<,x0<,kfun<,test>>>>>>>)

<alfa,sres,vres,yptr,yptt<,plan>> = svm(train,model,<,optn<,class<,ctun<,x0<,kfun<,test>>>>>>>)

Two methods for the tuning of the regularization parameter C and all kernel function parameters are provided. The `"tun"` option can have two string values, `"gsrch"` or `"bound"`,

"gsrch" a grid search for specified values of C and all kernel function parameters is performed. The `ctun` input argument must contain a $K \times nk$ matrix, where K is 1 (for C) plus the number of kernel parameters. Each row contains a number of nonzero values for which the search is performed. The first row always corresponds to values for C.

"bound" a bounded (pattern) optimization is performed inside a specifed (hyper-) rectangle. The `ctun` input argument must contain a $K \times 2$ matrix, where K is 1 (for C) plus the number of kernel parameters. The first column contains a lower and the second column an upper bound value restricting the rectangular search area. The currently used Nelder-Mead simplex algorithm, however, does not seem to be very successful and may soon be replaced by something more appropriate.

Note, since a linear kernel has no kernel function parameter, the search for a set of specified C values was already available in the last distributed version.
As an example, we use the 40 first observations of the Heart data set and the mean misclassiction error of 4-fold cross validation:

```
data = rspfile("..\\tdata\\heart_40.dat");
modl = "1 = 2 : 14";
class = 1;
```

Since we are specifying 5 grid values for `C` and 13 grid values for the parameter of the RBF kernel function we should have to evaluate 65 cross validation, each of them with 5 optimizations:

```
/* tuning by grid search: specify values */
c = [ .001 .1 1. 10. 100. ];
kp1 = [ .2 : .05 : .8 ];
tun_g = c |> kp1 ;


/*--- FQP: grid search tuning ---*/
optn = [ "print"           3 ,
         "popt"            1 ,
         "tun"       "gsrch" ,
         "fold"            4 ,
         "kern"       "rbf2" ,
         "kfp1"   .076923076,
         "meth"        "fqp" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun_g);
```

The function values are the mean misclassification across the 4 cross validations:

```
************************************
Results of Grid Search for 2 Parameters
************************************
```

N	F	C	KF1
1	12.7500000	0.0010000	0.2000000
2	12.7500000	0.0010000	0.2500000
3	12.7500000	0.0010000	0.3000000
4	12.7500000	0.0010000	0.3500000
5	12.7500000	0.0010000	0.4000000
6	12.7500000	0.0010000	0.4500000
7	12.7500000	0.0010000	0.5000000
8	12.7500000	0.0010000	0.5500000
9	12.7500000	0.0010000	0.6000000
10	12.7500000	0.0010000	0.6500000
11	12.7500000	0.0010000	0.7000000
12	12.7500000	0.0010000	0.7500000
13	9.00000000	0.0010000	0.8000000
14	4.00000000	0.1000000	0.2000000
15	4.50000000	0.1000000	0.2500000
16	5.00000000	0.1000000	0.3000000
17	5.25000000	0.1000000	0.3500000
18	6.25000000	0.1000000	0.4000000
19	6.25000000	0.1000000	0.4500000
20	6.00000000	0.1000000	0.5000000
21	6.00000000	0.1000000	0.5500000
22	6.00000000	0.1000000	0.6000000
23	5.75000000	0.1000000	0.6500000
24	5.50000000	0.1000000	0.7000000
25	5.50000000	0.1000000	0.7500000
26	5.50000000	0.1000000	0.8000000
27	1.75000000	1.0000000	0.2000000
28	1.25000000	1.0000000	0.2500000
29	1.00000000	1.0000000	0.3000000
30	1.00000000	1.0000000	0.3500000
31	0.75000000	1.0000000	0.4000000
32	0.75000000	1.0000000	0.4500000
33	0.75000000	1.0000000	0.5000000
34	0.50000000	1.0000000	0.5500000
35	0.25000000	1.0000000	0.6000000
36	0.25000000	1.0000000	0.6500000
37	0.00000000	1.0000000	0.7000000
38	0.00000000	1.0000000	0.7500000
39	0.00000000	1.0000000	0.8000000
40	4.75000000	10.000000	0.2000000
41	4.50000000	10.000000	0.2500000
42	3.75000000	10.000000	0.3000000
43	2.50000000	10.000000	0.3500000
44	4.25000000	10.000000	0.4000000
45	4.50000000	10.000000	0.4500000
46	4.25000000	10.000000	0.5000000
47	4.25000000	10.000000	0.5500000
48	3.25000000	10.000000	0.6000000
49	2.75000000	10.000000	0.6500000
50	2.25000000	10.000000	0.7000000
51	2.75000000	10.000000	0.7500000
52	2.75000000	10.000000	0.8000000
53	4.75000000	100.00000	0.2000000
54	5.25000000	100.00000	0.2500000
55	5.50000000	100.00000	0.3000000
56	4.75000000	100.00000	0.3500000
57	4.25000000	100.00000	0.4000000
58	6.00000000	100.00000	0.4500000
59	5.75000000	100.00000	0.5000000
60	7.50000000	100.00000	0.5500000
61	7.75000000	100.00000	0.6000000
62	7.25000000	100.00000	0.6500000
63	6.50000000	100.00000	0.7000000
64	9.00000000	100.00000	0.7500000
65	8.50000000	100.00000	0.8000000

Best 10 Results of Grid Search

N	F	C	KF1
37	0.00000000	1.0000000	0.7000000
38	0.00000000	1.0000000	0.7500000
39	0.00000000	1.0000000	0.8000000
36	0.25000000	1.0000000	0.6500000
35	0.25000000	1.0000000	0.6000000
34	0.50000000	1.0000000	0.5500000
33	0.75000000	1.0000000	0.5000000
32	0.75000000	1.0000000	0.4500000
31	0.75000000	1.0000000	0.4000000

```
                30  1.00000000  1.0000000  0.3500000
```

The following is the best solution we found:

```
 Parameter Estimates
 -------------------

  1 :           1      0.789          1          1     0.7079
  6 :       0.729     0.8606    0.02609     0.7443          1
 11 :      0.5914          1     0.2668          1      0.484
 16 :      0.6924          1          1     0.7629     0.7904
 21 :      0.7006     0.7577     0.5493     0.2186     0.7315
 26 :      0.2942     0.7267     0.4106     0.6511      0.727
 31 :      0.5767     0.9406     0.4672     0.9472          1
 36 :           1          1          1     0.4551     0.4144
```

We can verify the result by running a single optimization with the best parameters:

```
/*--- FQP: optimal parameters (C,KFP1) ---*/
optn = [ "print"           3 ,
         "popt"            1 ,
         "fold"            4 ,
         "c"              1. ,
         "kern"       "rbf2" ,
         "kfp1"           .8 ,
         "meth"        "fqp" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class);
```

The following input specifies the automatic search:

```
/* tuning by automatic NMS: specify bounds */
tun_s = [ 1.e-6   100. ,
          1.e-6     .5 ];


optn = [ "print"           3 ,
         "popt"            1 ,
         "tun"       "bound" ,
         "fold"            4 ,
         "kern"       "rbf2" ,
         "kfp1"   .076923076,
         "meth"        "fqp" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun_s);
```

The following shows a small automatic grid search for obtaining a good starting point for the Nelder-Mead algorithm:

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

```
Results of Grid Search for 2 Parameters
**************************************

     N           F          C        KF1
     1  12.7500000  1.00e-006  1.00e-006
     2  12.7500000  1.00e-006  0.0050010
     3  12.7500000  1.00e-006  0.0500009
     4  12.7500000  1.00e-006  0.5000000
     5  12.7500000  1.0000010  1.00e-006
     6  9.25000000  1.0000010  0.0050010
     7  4.25000000  1.0000010  0.0500009
     8  0.75000000  1.0000010  0.5000000
     9  12.7500000  10.000001  1.00e-006
    10  5.25000000  10.000001  0.0050010
    11  5.50000000  10.000001  0.0500009
    12  4.25000000  10.000001  0.5000000
    13  12.7500000  100.00000  1.00e-006
    14  6.50000000  100.00000  0.0050010
    15  7.75000000  100.00000  0.0500009
    16  5.75000000  100.00000  0.5000000
```

After that small grid search the Nelder-Mead algorithm is started using the best grid point as starting point.

```
 N. Variables             2
 Criterion       0.750000000
Iter rest  nfun act  optcrit  difcrit     std   delta     size
   1    0    15   1  0.50000  0.25000  0.1179  10.000  0.55469
```

The improvement from F=.75 to F=.5 is disappointing since we know already that F=0 exists:

```
  Tuning Result by Optimization F=0.5
  **********************************

Regularization C     0.7500010
First Kernel Par     0.5000000
```

If `ctun` is a

- scalar: a single regularization parameter `C` is specified. The `"tun"` option must not be specified.
- vector: If the `"tun"` option is not specified, only a grid search for the specified set of regularization parameters `C` is performed but no tuning of kernel function arguments.

Two new estimation methods were added:

NSVM this is the L_2 Newton method developed by Fung & Mangasarian [**?**]. This method can be applied to large problems with linear kernel. This algorithm is very fast and needs only a few iterations for convergence.

FSM this is the L_1 Newton method developed by Fung & Mangasarian [**?**]. The method is extrem useful for variable ("feature") selection and seems to be important for the analysis of data where $nvar >> nobs$ as

it is for micro array data in gene analysis. Even for many variables (up to 100,000) we can find a small subset of variables which predict a binary or multinomial target. The `FSM` estimation method may require the specification of two additional method specific options

`lines` an Armijo line search is performed,

`delta` initial value for inner grid search for the best choice of this parameter is performed.

`dfact` step factor for inner grid search for the best choice of this parameter is performed.

`dend` final value for inner grid search for the best choice of this parameter is performed.

Both, `NSVM` and `FSM` are implemented for linear and nonlinear kernel. However, the implementation for nonlinear kernel requires $nobs^2$ more double words memory.
For an illustration we use the data set Caco2 from Kristin Bennett's and Jinbo Bi's ([**?**]) website. It has 27 observations and 714 variables. We read two files: `caco_nam.dat` and `caco.dat` with the 714 variable names and all the data.

```
/* Caco: nobs= 27, nvar= 714 (Bi & Bennett) */
fo10 = " %16s %16s %16s %16s %16s %16s %16s %16s %16s %16s";
form = fo10;
for (j = 2; j <= 71; j++) form = strcat(form,fo10);
fo4 = " %16s %16s %16s %16s";
form = strcat(form,fo4);

fid = fopen("..\\tdata\\caco_nam.txt","r");
/* sscanf() is faster when nr and nc are specified: */
c_nam = fscanf(fid,form,.,1);
/* print c_nam; */


/*--- read data: 715 cols, first column is obs number ---*/
fid = fopen("..\\tdata\\caco.dat","r");
form = fo1 = " %g %g %g %g %g %g %g %g %g %g";
for (j = 2; j <= 71; j++) form = strcat(form,fo1);
fo5 = " %g %g %g %g %g";
form = strcat(form,fo5);
c_dat = fscanf(fid,form,27,.);
nr = nrow(c_dat); nc = ncol(c_dat);
print "Observations of c_dat.dat:",nr;
print "Columns of c_dat.dat:",nc;
cdat = c_dat[,2:715]; /* cut out col 1 */
```

Using the median of the response as cutoff value we create a binary response appropriate for our classification problem:

```
sopt = [ "ari" "std" "med" ];
mom = univar(cdat,sopt);
/* print mom; */
cutof = mom[3,714];
y = cdat[,714]; y = y .< cutof; cdat[,714] = y;
```

We now have 14 observation with zero and 13 observations with unity response. By specifying `delt=.001` and `dend=100.` we perform an inner grid search of the method specific parameter δ for the points $\delta = .001, .01, 1., 10., 100.$ A factor of 10 is assumed by default. Unfortunately this is sometimes needed for obtaining a convergence region. The following input executes the algorithm without line search:

```
/* FSM: without line search */
modl = "714 = 1 : 713";
class = 714;
optn = [ "print"           2 ,
         "pplan"             ,
         "meth"        "fsm" ,
         "delt"          .001 ,
         "dend"          100. ,
         "kern"       "line" ];
alfa = svm(cdat,modl,optn,class);
```

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

            Variable       Value    Nobs  Proportion
            Y[714]             0      14   51.851852
                               1      13   48.148148
```

The results indicate that $\delta = .1$ would be best and $\delta = 100$ does not converge during the maximum number of 400 iterations:

```
 [note] Set Regularization Parameter C=0.00157787

           ---Start Training Cycle: Technique=  FSM---

  FSM: delta=    0.01: niter=   39 gnorm=0.000162 crit= -0.122911
  FSM: delta=     0.1: niter=   10 gnorm=0.000477 crit= -0.122911
  FSM: delta=       1: niter=   13 gnorm=0.000954 crit= -0.122911
  FSM: delta=      10: niter=   63 gnorm=0.000932 crit= -0.122911
 [warning] A suboptimal solution with a relaxed convergence
           criterion 0.006463 is accepted for delta=100.
  FSM: delta=     100: niter=  400 gnorm=0.006463 crit=  -0.12289
           Solution for delta=0.01 is selected as best.
```

The model indicates 2 misclassifications which is unusual since it can easily be fit without any misclassification:

```
                    Classification Table
                    --------------------

                          |    Predicted
                 Observed |        0       1
                 ---------|-----------------
                        0 |       13       1
```

```
 1 |        1      12
```

The linear plane with $\gamma = 0.0647744$ has only 6 nonzero coefficients corresponding to the following variables:

```
Largest 6 Values
----------------

1      75 -0.199383340
2     687 -0.107564833
3     632  0.041827466
4     633  0.041827466
5     630 -0.041827289
6       1 -0.005628912
```

This is an excellent result of feature selection.

```
Total Number of Kernel Calls: 487
    Time for Optimization: 34
    Total Processing Time: 56
```

Using the `lines` option we execute the algorithm with Armijo line search:

```
/* FSM: with line search */
modl = "714 = 1 : 713";
class = 714;
optn = [ "print"           2 ,
         "pplan"             ,
         "meth"        "fsm" ,
         "delt"         .001 ,
         "dend"         100. ,
         "lines"             ,
         "kern"       "line" ];
alfa = svm(cdat,modl,optn,class);
```

The results indicate that $\delta = .1$ would be best and $\delta = 100$ does not converge during the maximum number of 400 iterations:

```
[note] Set Regularization Parameter C=0.00157787

          ---Start Training Cycle: Technique=  FSM---

 FSM: delta=    0.01: niter=   14 gnorm=0.000162 crit= -0.122911
 FSM: delta=     0.1: niter=   12 gnorm=0.000244 crit= -0.122911
 FSM: delta=       1: niter=   13 gnorm=0.000954 crit= -0.122911
 FSM: delta=      10: niter=   63 gnorm=0.000932 crit= -0.122911
[warning] A suboptimal solution with a relaxed convergence
          criterion 0.006463 is accepted for delta=100.
 FSM: delta=     100: niter=  400 gnorm=0.006463 crit=  -0.12289
          Solution for delta=0.01 is selected as best.
```

The linear plane has only 6 nonzero coefficients corresponding to the following variables:

```
                    Largest 6 Values
                    ----------------

                    1      75 -0.199383115
                    2     687 -0.107565035
                    3     632  0.041827348
                    4     633  0.041827348
                    5     630 -0.041827172
                    6       1 -0.005629567


            Total Number of Kernel Calls: 487
                Time for Optimization: 33
                Total Processing Time: 55
```

The L_2 Newton method NSVM converges fast but results in dense coefficient vectors:

```
/* NSVM: */
modl = "714 = 1 : 713";
class = 714;
optn = [ "print"           2 ,
         "pplan"             ,
         "meth"       "nsvm" ,
         "kern"       "line" ];
alfa = svm(cdat,modl,optn,class);
```

The algorithm converges after one iteration:

```
                  NSVM Iteration History

      iter        crit        cdif       gnorm        xdif
         1 -0.01142344  0.00985263  2.161e-015  0.00476242
```

with a perfect fit:

```
                    Classification Table
                    --------------------

                             |    Predicted
                    Observed |        0       1
                    ---------|-----------------
                           0 |       14       0
                           1 |        0      13
```

The computer time is obviously only for the long printed output (the linear plane with 713 values):

```
Total Number of Kernel Calls: 487
    Time for Optimization: 8
    Total Processing Time: 30
```

1.5 New Features: Multidimensional Arrays (Tensors)

This work is not completed yet. Hopefully, it will be completed at the next newsletter in September.

1.6 New Features: Uniform Random Generators

An additional number of uniform random generators mostly based on XOR-shift operation is implemented (Marsaglia, 2003).
The `rand(nr<,nc<,mtyp<,dist<,...>>>>)` function now permits additional specifications for uniform random generators:

Distr.	Add. Arg.	Description
"icmp"	a, b	uniform RNG, very bad 16 bit version in Watcom C Compiler, int version
"iuni"	a, b	uniform with lower range a and upper range b, int version Mooore, RAND Corporation, see Fishman, p. 605
"iacm"	a, b	uniform random generator by Schrage (1979) in ACM, int version this is not a good choice
"ikis"	a, b	uniform random generator KISS by Marsaglia & Tsang, int version
"iecu"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), int version
"imwc"	a, b	multiply-with-carry RNG (Marsaglia, 2003), period 2^{128}, int version
"ix128"	a, b	XOR RNG (Marsaglia, 2003), period 2^{128}, int version
"iwow"	a, b	modified XOR RNG (Marsaglia, 2003), period $2^{192} - 2^{32}$, int version
"dcmp"	a, b	uniform with lower range a and upper range b very bad 16 bit version in Watcom C Compiler, real version
"duni"	a, b	uniform with lower range a and upper range b Mooore, RAND Corporation, see Fishman, p. 605, real version
"dacm"	a, b	uniform random generator by Schrage (1979) in ACM, real version this is not a good choice
"dkis"	a, b	uniform random generator KISS by Marsaglia & Tsang, real version
"decu"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), real version
"dmwc"	a, b	multiply-with-carry RNG (Marsaglia, 2003), period 2^{128}, real version
"dx128"	a, b	XOR RNG (Marsaglia, 2003), period 2^{128}, real version
"dwow"	a, b	modified XOR RNG (Marsaglia, 2003), period $2^{192} - 2^{32}$, real version

The default is `"duni"`.
The following is a complete list of available uniform RNGs in CMAT:

List of Runtime Options (Contd.)	
Syntax	**Description**
RANDCMP	The RANDCMP option specifies that the uniform random of the host compiler is used where CMAT is compiled. The windows version of the Watcom C compiler is the worst choice (based on 16 bit).
RANDUNI	The RANDUNI option specifies that the uniform random generator developed by Moore from the RAND Corporation is used. The generator is described at Fishman, 1996, p.605. In SAS Language it is known as RANUNI. This is the default for uniform random numbers.
RANDACM	The RANDACM option specifies that the uniform random generator developed by Schrage in ACM TOMS (1979) is used. This is not a good choice.
RANDKISS	The RANDKISS option specifies that the uniform random generator KISS ("Keep It Simple Stupid") by Marsaglia & Tsang (2002) is used.
RANDLECU	The RANDLECU option specifies that the uniform random generator by L'Ecuyer (1999) is used.
RANDXOR32	The uniform RNG XOR by Marsaglia (2003) is used. Period $2^{32}-1$
RANDXOR64	The uniform RNG XOR64 by Marsaglia (2003) is used. This is the 64 bit version. Period: $2^{64}-1$
RANDXORWOW	The uniform RNG XORWOW by Marsaglia (2003) is used. This is probably the best choice available. Period: $2^{192}-2^{32}$
RANDXOR128	The uniform RNG XOR128 by Marsaglia (2003) is used. Period: $2^{128}-1$
RANDMWC3	The uniform RNG MWC by Marsaglia (2003) is used. This is a multiply-with-carry algorithm. Period: $2^{128}-1$

The default is RANDUNI. Note, you can check the quality of 32 bit integer uniform random generators using the `diehd(func,optn)` function which implements the *gcd*, *birthday spacing*, and *gorilla* tests from George Marsaglia's "diehard battery" of tests. For some test examples see `test trand.inp` and `test trand2.inp`.
The SAS RANUNI (Rand Corporation) algorithm shows the following behaviour w.r.t. the *gcd*, *birthday spacing*, and the *gorilla* test:

```
Steps of Euclid Algorithm:  p-value =  1.0000
Distribution of gcd Values: p-value =  0.6621
```

Unfortunately, there is a rather large difference between the observed and predicted values of the distribution resulting in a large p value:

```
 Expected vs. Observed Counts: D = (O-E)^2/E

      0     1     2     3     4     5     6     7     8     9  >=10
E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
O     5    38   104   269   466   671   754   730   642   495   826
D  81.9 294.3 539.4 512.9 267.1  15.6 104.2 627.7  1634  2780 15168

Sum(O-E)^2/E = 22025.0533, p-value =  1.0000
```

Also the p value for the Gorilla test seems to be very high:

```
          Gorilla test for 2^26 bits, Positions 0 to 31
             (e.g. takes ~20 minutes for 850MHz PC)

    Bits  0 to  7 1.000 0.019 0.000 0.039 0.006 0.484 0.007 0.779
    Bits  8 to 15 0.128 0.000 0.014 0.001 0.008 0.016 0.022 0.026
    Bits 16 to 23 0.249 0.000 0.151 0.011 0.170 0.000 0.050 0.036
    Bits 24 to 31 0.184 0.005 0.786 0.150 0.000 0.000 0.005 0.001
          KS test for the above 32 p values: p =  1.0000
```

The following are the test results for the KISS ("Keep It Simple Stupid") algorithm (Marsaglia, 2002):

```
          Steps of Euclid Algorithm:  p-value =  0.7184
          Distribution of gcd Values: p-value =  0.9135
```

This rng shows a beautiful match of observed with predicted values and a very small p value:

```
           Expected vs. Observed Counts: D = (O-E)^2/E

      0     1     2     3     4     5     6     7     8     9  >=10
E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
O    87   373   748   968   979   807   508   280   147    57    46
D   0.2   0.1   0.3   0.1   0.0   0.8   0.3   1.1   0.0   1.3   0.7

           Sum(O-E)^2/E =   4.9596, p-value =  0.1061
```

Also the Gorilla test indicates the good behavior of this rng:

```
          Gorilla test for 2^26 bits, Positions 0 to 31
             (e.g. takes ~20 minutes for 850MHz PC)

    Bits  0 to  7 0.243 0.682 0.942 0.192 0.274 0.153 0.974 0.818
    Bits  8 to 15 0.546 0.313 0.253 0.049 0.703 0.739 0.769 0.925
    Bits 16 to 23 0.718 0.414 0.179 0.999 0.151 0.116 0.732 0.058
    Bits 24 to 31 0.054 0.814 0.381 0.413 0.079 0.896 0.981 0.461
          KS test for the above 32 p values: p =  0.4985
```

The following are the test results for the MWC algorithm (Marsaglia, 2003):

```
          Steps of Euclid Algorithm:  p-value =  0.5250
          Distribution of gcd Values: p-value =  0.0828


           Expected vs. Observed Counts: D = (O-E)^2/E

      0     1     2     3     4     5     6     7     8     9  >=10
E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
O    87   370   704   963  1023   802   520   272   154    75    30
D   0.2   0.0   1.1   0.2   2.2   0.5   0.0   2.2   0.2   1.2   2.8
```

```
            Sum(O-E)^2/E =  10.6782, p-value =  0.6169
```

The following are the test results for the XOR128 algorithm (Marsaglia, 2003):

```
           Steps of Euclid Algorithm:  p-value =  0.1829
           Distribution of gcd Values: p-value =  0.7658


            Expected vs. Observed Counts: D = (O-E)^2/E

       0     1     2     3     4     5     6     7     8     9  >=10
E   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
O     78   403   752   942  1014   733   529   312   146    62    29
D    2.0   3.7   0.5   1.2   1.4   3.0   0.1   0.7   0.1   0.3   3.3

            Sum(O-E)^2/E =  16.3324, p-value =  0.9095


           Gorilla test for 2^26 bits, Positions 0 to 31
              (e.g. takes ~40 minutes for 850MHz PC)

    Bits  0 to  7 0.992 0.537 0.793 0.951 0.105 0.082 0.236 0.932
    Bits  8 to 15 0.752 0.929 0.900 0.547 0.444 0.541 0.333 0.714
    Bits 16 to 23 0.103 0.782 0.377 0.536 0.540 0.874 0.058 0.284
    Bits 24 to 31 0.315 0.956 0.748 0.770 0.737 0.967 0.317 0.016

          KS test for the above 32 p values: p =  0.7797
```

The following are the test results for the XORWOW algorithm (Marsaglia, 2003):

```
           Steps of Euclid Algorithm:  p-value =  0.5364
           Distribution of gcd Values: p-value =  0.7628


            Expected vs. Observed Counts: D = (O-E)^2/E

       0     1     2     3     4     5     6     7     8     9  >=10
E   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
O     95   334   727  1028   959   794   551   270   138    67    37
D    0.1   2.9   0.0   2.7   0.3   0.2   1.7   2.6   0.8   0.0   0.3

            Sum(O-E)^2/E =  11.6672, p-value =  0.6921


           Gorilla test for 2^26 bits, Positions 0 to 31
              (e.g. takes ~40 minutes for 850MHz PC)

    Bits  0 to  7 0.709 0.876 0.864 0.073 0.970 0.411 0.253 0.325
    Bits  8 to 15 0.638 0.522 0.767 0.392 0.628 0.973 0.465 0.432
```

```
Bits 16 to 23 0.666 0.997 0.694 0.674 0.539 0.590 0.699 0.084
Bits 24 to 31 0.477 0.564 0.022 0.815 0.806 0.527 0.992 0.436

      KS test for the above 32 p values: p =  0.9208
```

1.7 New Functions

New functions are implemented:

generead

factor

2 New Developments

New developments in CMAT are described in more detail in a corresponding `newdev.pdf` file on this site. Here only some short comments.

2.1 Function `generead`

a = generead("path",rstr<,optn<,code>>)

Sorry, this is not completed yet!

Purpose: This function reads files containing a standard for of micro array data into numeric objects.

Input:

Output:

Restrictions: 1.

Relationships:

Examples: 1. :

2.2 Function `factor`

< gof,est,resi,cov> = factor(data,optn<,wgt<,xini<,prior>>>)

Purpose: The `factor` function implements a number of exploratory factor analysis methods which can be used for ULS, ML, GLS, WLS, and DWLS estimation. The factor model

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

where $\mathbf{C}$ is a $n \times n$ covariance or correlation matrix, $\mathbf{L}$ is a *ntimesm* factor loading matrix, and $\mathbf{U} = diag$ is a $n \times n$ diagonal matrix of unique variances. It implements a number of popular orthogonal and oblique rotation methods and incorporates robust (nonnormal, Satorra-Bentler) goodness of fit indices, robust ASEs, and nonnormal (robust) standardized residuals for ULS, ML, and GLS estimates.

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

1. `nobs` by `nvar` matrix of raw data
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.

wgt This argument specifies the weight matrix for GLS, WLS, or DWLS estimation.

xini This argument specifies a starting vector for the parameter estimation.

prior This argument specifies a vector of prior communalities. This is valid only for methods where the unique variances are specified but not estimated. Communalities h_i are defined for the the correlation analysis as 1 minus the unique variances u_i.

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

Option Name	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
"asc"	string	type of asymptotic covariances
"bia"		biased: Browne's (1984) formula (3.4)
"unb"		unbiased: Browne's (1984) formula (3.8)
"cor"		Browne & Shapiro (1986) formula (3.2)
"cl"	string	type of confidence intervals
"none"		do not compute
"wald"		Wald linkelihood intervals (fast)
"plik"		profile likelihood intervals (slow)
"boot"		Bootstrap confidence intervals (slow)
"comp"		perform principal component analysis
		use only for exploratory FACTOR model
"data"	string	type of input data
"raw"		$N \times n$ raw data
"cor"		symmetric correlation matrix
"cov"		symmetric covariance matrix
"dfred"	real	reduce degrees of freedom of χ^2 test
"demph"	real	add positive constant to initial values
"evsing"	real	threshhold for identifying zero eigenvalues
"freq"	int	column number of FREQ variable when raw data input
"frot"	string	rotation method for exploratory FACTOR model
	"non"	no rotation
	"qua"	quartimax rotation
	"var"	varimax rotation
	"equ"	equamax rotation
	"par"	parsimax rotation
	"pri"	principal component rotation
"fnorm"		Kaiser's factor pattern normalization
		use only for exploratory FACTOR model
"hey"		Heywood case: apply boundary constraints
"inic"		compute initial values
"met"	string	estimation method
	"no"	no estimation, just evaluate model
	"ml"	maximum likelihood (ML)
	"ls"	unweighted least squares (LS)
	"gls"	generalized least squares (GLS)
	"wls"	weighted least squares (WLS,ADF)
	"dwls"	diagonally weighted LS (DWLS, app. ADF)
"maxfu"	int	maximum nuber function calls for optimization
"maxit"	int	maximum number iterations for optimization
"nfac"	real	number factors for exploratory FACTOR model
"nobs"	real	number observations when COV or CORR data input
		only for single sample analysis
"nopr"		perform no printed output

Option Name	Second Column	Meaning
"prin"	int	medium amount of printed output
"pall"		large amount of printed output
"psho"		short amount of printed output
"psum"		summary amount of printed output
"rob"		robust estimation of ASE and χ^2
"ridge"	real	nonnegative ridge value
"rtres"	real	small threshold for setting residuals to zero
"start"	string	type of residual matrix
	"com"	input communalities
	"efa"	LS EFA by linear algebra
	"ran"	pseudo random
"sing"	real	singularity threshold (default: 1.e-8)
"seed"	real	seed value for random generator
"tech"	string	optimization technique
	"tr"	trust region
	"lm"	Levenberg-Marquardt
	"cg"	conjugate gradient
	"nrl"	Newton-Raphson line search
	"nrr"	Newton-Raphson ridged
	"dqn"	dual Quasi Newton
	"nms"	Nelder-Mead simplex
	"qpn"	quadratic penalty algorithm (NLC)
	"sqp"	sequential QP algorithm (NLC)
	"cb"	Cobyla (NLC)
"wgt"	int	column number of WEIGHT variable when raw data input
"wridge"	real	ridge factor for weight matrix (GLS, WLS, DWLS)
"wpen"	real	penalty weight for WLS and DWLS estimation

Output: gof column vector with goodness-of-fit measures.

parm vector or matrix with parameter estimates and if the `cl` option is specified specified also with asymptotic standard errors (ASEs) and confidence intervals.

- `cl` is not specified: `parm` contains the $n \times m$ matrix $\mathbf{L}$ of factor loadings.
- `cl` is specified: The first column contains all the parameter estimates, i.e. the entries of $\mathbf{L}$ and $\mathbf{U}$. The following four columns contains normal theory ASE, t value, lower and upper Wald confidence limits. If raw data are given and other than WLS estimates are specified an additional set of four columns contain the robust (nonnormal theory, Satorra-Bentler) ASE, t value, lower and upper confidence limits.

resi matrix of predicted values and residuals. There are $n(n+1)/2$ rows corresponding to entries $(i, j), i \geq j$ of the lower triangle of a symmetric matrix and 6 or 7 columns specifying:

1. analyzed CORR or COV data ($\mathbf{S}$)
2. model matrix $\mathbf{\Sigma}$
3. raw residuals $r_{ij} = s_{ij} - \sigma_{ij}$
4. normalized residuals
5. standardized residuals
6. normal theory asymptotically standardized residuals
7. only if raw data are given: robust asymptotically standardized residuals

cov the result depends whether robust estimates are computed (raw data are available) or not:

- without robust estimation: `cov` contains the $p \times p$ covariance matrix of parameter estimates.
- with robust estimation: The first column of `cov` contains all $p(p+1)/2$ lower triangular entries of the normal theory covariances; the second columns contains the corresponding entries of the robust covariance matrix.

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

2.

Relationships: noharm(), sem(), rotate()

Examples: LOT Example, nobs=389, nvar=8:

1. ML Estimates:

```
  options NOECHO;
  lot8 = [
#include "..\\tdata\\lot.dat"
           ];
  options ECHO;

  lot = shape(lot8,.,8);
  cnam = [ "i1" "i4" "i5" "i11" "i3" "i8" "i9" "i12" ];
  lot = cname(lot,cnam);
  print cov = bivar(lot,"cov");


  /* SEM: ML Varimax: COV Analysis */
  optn = [ "data"      "raw" ,
           "anal"      "cov" ,
           "meth"       "ml" ,
           "vers"          1 ,
           "nfac"          2 ,
           "tech"   "trureg" ,
           "frot"     "vmax" ,
           "prin"          3 ];
  gof = factor(lot,optn);
```

```
                ***********************************
                Maximum Likelihood Factor Analysis
                  (Exploratory Factor Analysis)
                ***********************************

  Number of Items . . . . . . . . . . . . . . . .         8
  Specified Number of Factors . . . . . . . . . .         2
  Number of Subjects. . . . . . . . . . . . . . .       389
  Version . . . . . . . . . . . . . . . SEM-Type Matrix Model
  Rotation Method . . . . . . . . . . . . . . . .   VARIMAX


  Name             Mean     Std Dev    Skewness    Kurtosis
  ------------------------------------------------------------
  i1        2.244215938  1.00744584 -0.12460927 -0.64820145
```

```
i4        2.439588689  0.99201754 -0.34919434 -0.35531466
i5        2.596401028  0.98916158 -0.56683881 -0.11488129
i11       2.336760925  0.98557158 -0.56994653 -0.09588667
i3        1.845758355  1.05364549  0.24526977 -0.72410856
i8        1.393316195  1.03149204  0.62842416 -0.14311245
i9        1.321336761  1.00107276  0.68497038  0.00779773
i12       1.398457584  1.06881652  0.70560377 -0.23352418
```

```
Mardia's Multivariate Kurtosis . . . . . .     17.8826
Relative Multivariate Kurtosis . . . . . .      1.2235
Normalized Multivariate Kurtosis . . . . . .    13.9417
Mardia Based Kappa (Browne, 1982). . . . . .     0.2235
Mean Scaled Univariate Kurtosis  . . . . . .    -0.0961
Adjusted Mean Scaled Univariate Kurtosis . . .  -0.0890
Multivariate Mean Kappa (Bentler, 1985). . . .   0.0000
Multivariate LS Kappa (Bentler, 1983). . . . .   0.0000
```

```
Observation numbers with largest contribution to kurtosis


          294        94       376       381       335
      1750.762  900.0132  836.6881  739.1942  525.3757
```

Covariance Matrix

```
             i1         i4         i5        i11         i3
i1    1.0149471
i4    0.5109254  0.9840988
i5    0.4390255  0.5232356  0.9784406
i11   0.2500199  0.3361183  0.2187409  0.9713513
i3   -0.1658429 -0.2335688 -0.2660205 -0.1128720  1.1101688
i8   -0.2896006 -0.3846832 -0.3382715 -0.1972279  0.5479355
i9   -0.2410489 -0.2885273 -0.2978096 -0.2193173  0.5368047
i12  -0.2315811 -0.3663305 -0.3129953 -0.2685514  0.4920295
```

Covariance Matrix

```
                i8         i9        i12
i8    1.0639758
i9    0.7238028  1.0021467
i12   0.5928763  0.5546206  1.1423687
Determinant =  9.87308500e-002 (Ln = -2.3154e+000)
```

```
Trust Region Optimization
Without Parameter Scaling
User Specified Gradient
User Specified Hessian (dense)
```

```
 Iteration Start:
 N. Variables             23
 Criterion       0.065924544        Max Grad Entry  0.150498090
 TR Radius       1.000000000

 Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius

   1    0    2   0  0.045703 0.020221 0.02559 0.00000  1.00000
   2    0    3   0  0.045148 5.6e-004  3e-003 0.00000  0.15722
   3    0    4   0  0.045125 2.3e-005  8e-004 0.00000  0.03310
   4    0    5   0  0.045124 1.3e-006  2e-004 0.00000   8e-003
   5    0    6   0  0.045124 7.8e-008  5e-005 0.00000   2e-003
   6    0    7   0  0.045124 4.8e-009  1e-005 0.00000   5e-004
   7    0    8   0  0.045124 2.9e-010  3e-006 0.00000   1e-004

 Successful Termination After     7 Iterations
 GCONV convergence criterion satisfied.
 Criterion       0.045124066        Max Grad Entry  2.9980e-006
 Ridge (lambda)  0.000000000        TR Radius       0.000103904
 Act.dF/Pred.dF  1.217754783
 N. Function Calls         9        N. Gradient Calls         5
 N. Hessian Calls          9        Preproces. Time           0
 Time for Method           1        Effective Time            1

[note] Convergence criterion satisfied.

                    Predicted Model Matrix

              i1         i4         i5        i11         i3
 i1    1.0149471
 i4    0.5197183  0.9840988
 i5    0.4094512  0.5261132  0.9784406
 i11   0.2498831  0.3213048  0.2585538  0.9713513
 i3   -0.1808276 -0.2384015 -0.2320363 -0.1534957  1.1101688
 i8   -0.2859003 -0.3748389 -0.3509231 -0.2291148  0.5583732
 i9   -0.2272029 -0.2997740 -0.2933122 -0.1943662  0.5440119
 i12  -0.2727516 -0.3559182 -0.3219539 -0.2076546  0.4435728

                    Predicted Model Matrix

                         i8         i9        i12
            i8    1.0639758
            i9    0.7170171  1.0021467
            i12   0.5972610  0.5688698  1.1423687
      Determinant =  1.00000000e+000 (Ln =  0.0000e+000)

                       Residual Matrix
```

```
             i1         i4         i5         i11         i3
i1    0.0000000
i4   -0.0087928  0.0000000
i5    0.0295743 -0.0028776  0.0000000
i11   1.37e-004  0.0148134 -0.0398129  0.0000000
i3    0.0149848  0.0048328 -0.0339842  0.0406237  0.0000000
i8   -0.0037003 -0.0098442  0.0126515  0.0318869 -0.0104377
i9   -0.0138460  0.0112467 -0.0044974 -0.0249511 -0.0072073
i12   0.0411705 -0.0104123  0.0089586 -0.0608968  0.0484567


                    Residual Matrix

                         i8         i9         i12
           i8    0.0000000
           i9    0.0067857  0.0000000
           i12  -0.0043848 -0.0142492  0.0000000

      Average Off-diagonal Residual = 1.84291783e-002
                 Sum-of-Squares= 0.0164356


           Asymptotically Standardized Residuals

             i1         i4         i5         i11         i3
i1    0.0000000
i4   -1.5667407  0.0000000
i5    1.9608334 -0.5190397  0.0000000
i11   0.0045809  1.0471200 -1.3848371  0.0000000
i3    0.5373738  0.3357607 -1.2679678  1.1621694  0.0000000
i8   -0.2309920 -1.1947884  0.8263281  1.5914559 -0.7930526
i9   -0.9381785  1.4956615 -0.3151845 -1.3353879 -0.6020630
i12   1.5126515 -0.7484530  0.3439445 -1.7802064  1.6484049


           Asymptotically Standardized Residuals

                         i8         i9         i12
           i8    0.0000000
           i9    1.8333694  0.0000000
           i12  -0.3258078 -1.1514432  0.0000000

      Average Off-diagonal Residual = 1.01620703e+000
                  Sum-of-Squares= 37.2769


   Loadings and Unique Variances with ASE's and Wald CIs
   -----------------------------------------------------

                        FAC_1                 FAC_2
     i1   -0.6365460  0.0538743  0.0000000  0.0000000
          [-0.742138,-0.530954] [ 0.000000, 0.000000]
```

```
i4   -0.8164661  0.0535457 -0.0110432  0.0738916
     [-0.921414,-0.711519] [-0.155868, 0.133782]

i5   -0.6432389  0.0521942 -0.0842522  0.0644841
     [-0.745538,-0.540940] [-0.210639, 0.042134]

i11  -0.3925609  0.0546233 -0.0717292  0.0599655
     [-0.499621,-0.285501] [-0.189259, 0.045801]

i3    0.2840763  0.0711653  0.5852352  0.0575131
     [ 0.144595, 0.423558] [ 0.472512, 0.697959]

i8    0.4491432  0.0745767  0.7360840  0.0564318
     [ 0.302976, 0.595311] [ 0.625480, 0.846688]

i9    0.3569309  0.0752445  0.7563050  0.0532881
     [ 0.209454, 0.504407] [ 0.651862, 0.860748]

i12   0.4284869  0.0690507  0.5499496  0.0604561
     [ 0.293150, 0.563824] [ 0.431458, 0.668441]
```

```
                          U_Var
          i1    0.6097563  0.0568093
               [ 0.498412, 0.721100]

          i4    0.3173599  0.0601064
               [ 0.199553, 0.435166]

          i5    0.5575859  0.0512516
               [ 0.457135, 0.658037]

          i11   0.8121022  0.0610978
               [ 0.692353, 0.931852]

          i3    0.6869692  0.0557412
               [ 0.577718, 0.796220]

          i8    0.3204266  0.0403661
               [ 0.241311, 0.399543]

          i9    0.3027498  0.0415678
               [ 0.221278, 0.384221]

          i12   0.6563232  0.0528909
               [ 0.552659, 0.759987]
```

Loadings and Unique Variances with Robust ASE's

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

                    FAC_1                 FAC_2
i1   -0.6365460  0.0529160  0.0000000  0.0000000
     [-0.740260,-0.532833] [ 0.000000, 0.000000]

i4   -0.8164661  0.0509288 -0.0110432  0.0798594
     [-0.916285,-0.716647] [-0.167565, 0.145478]

i5   -0.6432389  0.0571137 -0.0842522  0.0735942
     [-0.755180,-0.531298] [-0.228494, 0.059990]

i11  -0.3925609  0.0601718 -0.0717292  0.0660081
     [-0.510496,-0.274626] [-0.201103, 0.057644]

i3    0.2840763  0.0785461  0.5852352  0.0610253
     [ 0.130129, 0.438024] [ 0.465628, 0.704843]

i8    0.4491432  0.0791311  0.7360840  0.0612133
     [ 0.294049, 0.604237] [ 0.616108, 0.856060]

i9    0.3569309  0.0833071  0.7563050  0.0545688
     [ 0.193652, 0.520210] [ 0.649352, 0.863258]

i12   0.4284869  0.0837765  0.5499496  0.0662955
     [ 0.264288, 0.592686] [ 0.420013, 0.679886]


                         U_Var
           i1    0.6097563  0.0604097
                [ 0.491355, 0.728157]

           i4    0.3173599  0.0602473
                [ 0.199277, 0.435442]

           i5    0.5575859  0.0613533
                [ 0.437336, 0.677836]

           i11   0.8121022  0.0622507
                [ 0.690093, 0.934111]

           i3    0.6869692  0.0734604
                [ 0.542990, 0.830949]

           i8    0.3204266  0.0606124
                [ 0.201628, 0.439225]

           i9    0.3027498  0.0482205
                [ 0.208239, 0.397260]

           i12   0.6563232  0.0824520
```

```
[ 0.494720, 0.817926]

(1) Standalone Fit Measures: ------------------------------
Fit criterion . . . . . .     0.0451
Normal Th. Chi-square (df = 13) 17.5081 Prob>chi**2 = 0.1771
Satorra-Bentler Mean (df= 13) . 14.6707 Prob>chi**2 = 0.3284
Sat.-B. M-V (df= 9.703) . . . . 10.9496 Prob>chi**2 = 0.3364
Elliptic cor. Chi-square. . . . 14.3095 Prob>chi**2 = 0.3524
Normal Theory Reweighted LS Chi-square  . . . . .   191.8518
Probability of Close Fit  . . . . . .     0.8222
Z-Test of Wilson & Hilferty (1931). . . . . .     0.9287
(2) Incremental Fit Measures: ------------------------------
Null Model Chi-square (df = 28) . . . . . .     995.2496
RMSEA Estimate  . . . . .  0.0299  90%C.I.[ 0.0000,  0.0623]
ECVI Estimate . . . . . .  0.1665  90%C.I.[ 0.0000,  0.2058]
McDonald's (1989) Centrality. . . . . .     0.9942
Tucker-Lewis Coefficient TLI. . . . . .     0.9900
Bentler & Bonett's (1980) NFI . . . . . .     0.9824
Bentler's Comparative Fit Index . . . . . .     0.9953
Parsimonious NFI (James, Mulaik, & Brett,1982). .     0.4561
Bollen (1986) Normed Index Rho1 . . . . . .     0.9621
Bollen (1988) Non-normed Index Delta2 . . . . . .     0.9954
(3) Information Criteria: ---------------------------------
Akaike's Information Criterion. . . . . .    -8.4919
Bozdogan's (1987) CAIC. . . . . .   -73.0184
Schwarz's Bayesian Criterion. . . . . .   -60.0184
(4) Other Fit Measures: -----------------------------------
Goodness of Fit Index (GFI) . . . . . .     0.9889
Parsimonious GFI (Mulaik, 1989) . . . . . .     0.4591
GFI Adjusted for Degrees of Freedom (AGFI). . . .     0.9693
Root Mean Square Residual (RMR) . . . . . .     0.0214
Hoelter's (1983) Critical N . . . . . .        497
```

VARIMAX Rotated Factor Loadings

	FAC_1	FAC_2
i1	0.6127570	-0.1723939
i4	0.7829624	-0.2317516
i5	0.5963820	-0.2553100
i11	0.3584639	-0.1753646
i3	-0.1149623	0.6402994
i8	-0.2330063	0.8302152
i9	-0.1387638	0.8247069
i12	-0.2635322	0.6454427

Orthogonal Transformation Matrix

	FAC_1	FAC_2
FAC_1	-0.9626280	0.2708271

```
FAC_2   0.2708271  0.9626280
```

2. ULS Estimates:

```
/* SEM: ULS Varimax: COV Analysis */
optn = [ "data"      "raw" ,
         "anal"      "cov" ,
         "meth"      "uls" ,
         "cl"            1 ,
         "vers"          1 ,
         "nfac"          2 ,
         "tech"   "trureg" ,
         "frot"     "vmax" ,
         "prin"          3 ];
gof = factor(lot,optn);
```

```
*****************************
Least Squares Factor Analysis
(Exploratory Factor Analysis)
*****************************

Number of Items . . . . . .          8
Specified Number of Factors .          2
Number of Subjects. . . . . .        389
Version . . . . . SEM-Type Matrix Model
Rotation Method . . . . . .    VARIMAX

Name             Mean     Std Dev    Skewness    Kurtosis
-----------------------------------------------------------
i1         2.244215938  1.00744584 -0.12460927 -0.64820145
i4         2.439588689  0.99201754 -0.34919434 -0.35531466
i5         2.596401028  0.98916158 -0.56683881 -0.11488129
i11        2.336760925  0.98557158 -0.56994653 -0.09588667
i3         1.845758355  1.05364549  0.24526977 -0.72410856
i8         1.393316195  1.03149204  0.62842416 -0.14311245
i9         1.321336761  1.00107276  0.68497038  0.00779773
i12        1.398457584  1.06881652  0.70560377 -0.23352418

Mardia's Multivariate Kurtosis . . . . . .     17.8826
Relative Multivariate Kurtosis . . . . . .      1.2235
Normalized Multivariate Kurtosis . . . . . .     13.9417
Mardia Based Kappa (Browne, 1982). . . . . .      0.2235
Mean Scaled Univariate Kurtosis  . . . . . .     -0.0961
Adjusted Mean Scaled Univariate Kurtosis . . .     -0.0890
Multivariate Mean Kappa (Bentler, 1985). . . .      0.0000
Multivariate LS Kappa (Bentler, 1983). . . . .      0.0000

Observation numbers with largest contribution to kurtosis
```

```
            294        94       376       381       335
       1750.762  900.0132  836.6881  739.1942  525.3757


                    Covariance Matrix

             i1         i4         i5        i11         i3
i1    1.0149471
i4    0.5109254  0.9840988
i5    0.4390255  0.5232356  0.9784406
i11   0.2500199  0.3361183  0.2187409  0.9713513
i3   -0.1658429 -0.2335688 -0.2660205 -0.1128720  1.1101688
i8   -0.2896006 -0.3846832 -0.3382715 -0.1972279  0.5479355
i9   -0.2410489 -0.2885273 -0.2978096 -0.2193173  0.5368047
i12  -0.2315811 -0.3663305 -0.3129953 -0.2685514  0.4920295


                    Covariance Matrix

                         i8         i9        i12
           i8    1.0639758
           i9    0.7238028  1.0021467
           i12   0.5928763  0.5546206  1.1423687
    Determinant =  9.87308500e-002 (Ln = -2.3154e+000)


                  Trust Region Optimization
                  Without Parameter Scaling
                   User Specified Gradient
                User Specified Hessian (dense)


 Iteration Start:
 N. Variables             23
 Criterion       0.020166120        Max Grad Entry  0.093355791
 TR Radius       1.000000000

Iter rest nfun act   optcrit  difcrit maxgrad  lambda   radius

   1    0    2   0  0.015472 4.7e-003  6e-003 0.00000  1.00000
   2    0    3   0  0.015414 5.7e-005  8e-004 0.00000  0.13604
   3    0    4   0  0.015414 8.4e-007  1e-004 0.00000  0.01776
   4    0    5   0  0.015414 1.9e-008  2e-005 0.00000   2e-003
   5    0    6   0  0.015414 4.5e-010  3e-006 0.00000   3e-004

 Successful Termination After     5 Iterations
 ABSGCONV convergence criterion satisfied.
 Criterion       0.015413570        Max Grad Entry  3.3883e-006
 Ridge (lambda)  0.000000000        TR Radius       0.000308874
 Act.dF/Pred.dF  0.939876329
```

```
  N. Function Calls         7        N. Gradient Calls         5
  N. Hessian Calls          7        Preproces. Time           0
  Time for Method           0        Effective Time            0

[note] Convergence criterion satisfied.
```

Predicted Model Matrix

	i1	i4	i5	i11	i3
i1	1.0149471				
i4	0.5251075	0.9840988			
i5	0.4137600	0.5236769	0.9784406		
i11	0.2497837	0.3169036	0.2561781	0.9713513	
i3	-0.1726696	-0.2378846	-0.2338023	-0.1546476	1.1101688
i8	-0.2760594	-0.3723567	-0.3497857	-0.2275859	0.5618310
i9	-0.2248128	-0.3082732	-0.3000414	-0.1977742	0.5466064
i12	-0.2664000	-0.3547363	-0.3237078	-0.2082913	0.4610163

Predicted Model Matrix

	i8	i9	i12
i8	1.0639758		
i9	0.7003591	1.0021467	
i12	0.6028354	0.5758332	1.1423687

Determinant = 1.00000000e+000 (Ln = 0.0000e+000)

Residual Matrix

	i1	i4	i5	i11	i3
i1	0.0000000				
i4	-0.0141820	0.0000000			
i5	0.0252655	-4.41e-004	0.0000000		
i11	2.36e-004	0.0192147	-0.0374372	0.0000000	
i3	0.0068267	0.0043158	-0.0322181	0.0417756	0.0000000
i8	-0.0135412	-0.0123265	0.0115141	0.0303580	-0.0138955
i9	-0.0162362	0.0197459	0.0022318	-0.0215431	-0.0098018
i12	0.0348189	-0.0115942	0.0107126	-0.0602601	0.0310133

Residual Matrix

	i8	i9	i12
i8	0.0000000		
i9	0.0234437	0.0000000	
i12	-0.0099591	-0.0212126	0.0000000

Loadings and Unique Variances with ASE's and Wald CIs

```
                    FAC_1                FAC_2
i1   -0.6463151  0.0572119  0.0000000  0.0000000
     [-0.758448,-0.534182] [ 0.000000, 0.000000]

i4   -0.8124636  0.0688247 -0.0343266  0.0921445
     [-0.947358,-0.677570] [-0.214926, 0.146273]

i5   -0.6401831  0.0539660 -0.1034589  0.0708719
     [-0.745955,-0.534412] [-0.242365, 0.035447]

i11  -0.3864735  0.0449466 -0.0847128  0.0552407
     [-0.474567,-0.298380] [-0.192983, 0.023557]

i3    0.2671601  0.0601181  0.6067235  0.0569453
     [ 0.149331, 0.384989] [ 0.495113, 0.718334]

i8    0.4271281  0.0675108  0.7379299  0.0658470
     [ 0.294809, 0.559447] [ 0.608872, 0.866988]

i9    0.3478377  0.0678574  0.7477510  0.0647232
     [ 0.214840, 0.480836] [ 0.620896, 0.874606]

i12   0.4121829  0.0590985  0.5783482  0.0593819
     [ 0.296352, 0.528014] [ 0.461962, 0.694735]


                         U_Var
          i1    0.5972239  0.1030719
               [ 0.395207, 0.799241]

          i4    0.3228234  0.1295580
               [ 0.068894, 0.576752]

          i5    0.5579025  0.0949086
               [ 0.371885, 0.743920]

          i11   0.8148133  0.0780252
               [ 0.661887, 0.967740]

          i3    0.6706809  0.0917567
               [ 0.490841, 0.850521]

          i8    0.3369968  0.1049889
               [ 0.131222, 0.542771]

          i9    0.3220241  0.1074089
               [ 0.111506, 0.532542]

          i12   0.6379874  0.0889375
               [ 0.463673, 0.812302]
```

S-B Adjusted Residual Matrix

```
             i1         i4         i5        i11         i3
i1    0.0000000
i4    0.2513180  0.0000000
i5   -0.4392446  0.0069001  0.0000000
i11  -0.0033773 -0.3161153  0.5314251  0.0000000
i3   -0.0943013 -0.0638424  0.4311451 -0.5262197  0.0000000
i8    0.2064289  0.1751678 -0.1649490 -0.4508588  0.1962302
i9    0.2563607 -0.2997725 -0.0331519  0.2884885  0.1478092
i12  -0.4546288  0.1555483 -0.1367382  0.7923820 -0.3742028
```

S-B Adjusted Residual Matrix

```
             i8         i9        i12
i8    0.0000000
i9   -0.3140747  0.0000000
i12   0.1241752  0.3004554  0.0000000
```

Loadings and Unique Variances with Robust ASE's

```
                    FAC_1                 FAC_2
i1   -0.6463151  0.0542930  0.0000000  0.0000000
     [-0.752727,-0.539903] [ 0.000000, 0.000000]

i4   -0.8124636  0.0521477 -0.0343266  0.0826977
     [-0.914671,-0.710256] [-0.196411, 0.127758]

i5   -0.6401831  0.0577076 -0.1034589  0.0743294
     [-0.753288,-0.527078] [-0.249142, 0.042224]

i11  -0.3864735  0.0608677 -0.0847128  0.0662385
     [-0.505772,-0.267175] [-0.214538, 0.045112]

i3    0.2671601  0.0809388  0.6067235  0.0599960
     [ 0.108523, 0.425797] [ 0.489134, 0.724313]

i8    0.4271281  0.0799420  0.7379299  0.0597386
     [ 0.270445, 0.583812] [ 0.620844, 0.855015]

i9    0.3478377  0.0834387  0.7477510  0.0531659
     [ 0.184301, 0.511375] [ 0.643548, 0.851954]

i12   0.4121829  0.0858137  0.5783482  0.0651318
     [ 0.243991, 0.580375] [ 0.450692, 0.706004]
```

```
                    U_Var
i1    0.5972239  0.0625506
     [ 0.474627, 0.719821]

i4    0.3228234  0.0616402
     [ 0.202011, 0.443636]

i5    0.5579025  0.0618747
     [ 0.436630, 0.679175]

i11   0.8148133  0.0628124
     [ 0.691703, 0.937923]

i3    0.6706809  0.0729067
     [ 0.527786, 0.813575]

i8    0.3369968  0.0606754
     [ 0.218075, 0.455918]

i9    0.3220241  0.0482885
     [ 0.227380, 0.416668]

i12   0.6379874  0.0798265
     [ 0.481530, 0.794444]
```

```
(1) Standalone Fit Measures: -------------------------------
Fit criterion . . . . . .     0.0154
Normal Th. Chi-square (df = 13)  5.9805 Prob>chi**2 = 0.9469
Satorra-Bentler Mean (df= 13) . 17.2141 Prob>chi**2 = 0.1897
Sat.-B. M-V (df= 10.28) . . . . 13.6184 Prob>chi**2 = 0.2084
(2) Incremental Fit Measures: ------------------------------
Null Model Chi-square (df = 28) . . . . . .    1610.6568
RMSEA Estimate  . . . . .  0.0000  90%C.I.[ 0.0000,  0.0062]
ECVI Estimate . . . . . .  0.1340  90%C.I.[ 0.0000,  0.1526]
(3) Information Criteria: ----------------------------------
Akaike's Information Criterion. . . . . .   -20.0195
Bozdogan's (1987) CAIC. . . . . .   -84.5461
Schwarz's Bayesian Criterion. . . . . .   -71.5461
(4) Other Fit Measures: ------------------------------------
Goodness of Fit Index (GFI) . . . . . .     0.9982
Parsimonious GFI (Mulaik, 1989) . . . . . .     0.4634
GFI Adjusted for Degrees of Freedom (AGFI). . . .     0.9949
Root Mean Square Residual (RMR) . . . . . .     0.0207
```

VARIMAX Rotated Factor Loadings

```
          FAC_1      FAC_2
  i1    0.6258566 -0.1613280
  i4    0.7781775 -0.2360406
```

```
i5    0.5940941 -0.2599814
i11   0.3530948 -0.1784997
i3   -0.1072580  0.6542046
i8   -0.2294117  0.8211877
i9   -0.1501797  0.8109061
i12  -0.2547730  0.6629269


Orthogonal Transformation Matrix

             FAC_1      FAC_2
FAC_1  -0.9683459  0.2496120
FAC_2   0.2496120  0.9683459
```

3 Illustration

3.1 Reading String Data and Type Conversion

The German data set from the UCI Repository website contains 1000 observations (rows) with 21 variables (columns). For an example the following is the first observation: response on the end:

```
A11 6 A34 A43 1169 A65 A75 4 A93 A101
4 A121 67 A143 A152 2 A173 1 A192 A201 1 */
```

```
/* German: nobs=1000, nvar=20 */
fo1 = " %s %d %s %s %d %s %s %d %s %s";
fo2 = " %d %s %d %s %s %d %s %d %s %s %d";
form = strcat(fo1,fo2);
fid = fopen("..\\tdata\\german.dat","r");
/* sscanf() is faster when nr and nc are specified: */
data = fscanf(fid,form,1000,21);
/* print data; */
nr = nrow(data); nc = ncol(data);
print "Observations of german.dat:",nr;
```

The first column contains the values of a char variable which always has the prefix `A1` and the one-digit suffix indicating a category. In the following we illustrate how to cut out the numeric suffix by first seprating the first column in vector `col1`:

```
col1 = data[,1]; /* print col1; */
```

Now we can use the `sscanf` function for separating each 3 char strings into a 2 char string (stored into `ind[1]`) and an integer value (stored into `ind[2]`).:

```
/* sscanf version */
stmp = " "; ind = 0;
num = cons(nr,1,1);
for (i = 1; i <= nr; i++) {
```

```
  str = col1[i]; ind = sscanf(str,"%2s %1d",1,2);
  num[i] = ind[2];
}
```

Remember, we can access the characters of a string scalar using indices. The `(int)` cast will then perform the type conversion on the third character of the string in `str`:

```
/* cast to int is now working */
num = cons(nr,1,1);
for (i = 1; i <= nr; i++) {
  str = col1[i]; num[i] = (int)str[3];
}
```

In a similar way you can use the `sprintf` option for converting interger and real data into strings.

3.2 Notes on Comparison Operations

In CMAT, the conditional expression can be used to generate $(0, 1)$ valued objects. The following operators can be used for elementwise comparison either between two objects of the same dimension or between an object and a scalar.

`.<`	elementwise less-than relation
`.>`	elementwise greater-than relation
`.<=`	elementwise less-than-or-equal-to relation
`.>=`	elementwise greater-than-or-equal-to relation
`.==`	elementwise equal-to relation
`.!=`	elementwise not-equal-to relation

The result is a binary object (with values 0 or 1) of the same dimension. In addition, you could then use the `replace` function for assigning other values.
Using these operators can save considerable computing time compared with the use of `for` or `while` loops, especially for large objects. For example, the following CMAT code illustrates scoring a logistic model.

```
remis = [ 1   .8   .83  .66  1.9  1.1     .996 ,
          1   .9   .36  .32  1.4   .74    .992 ,
          0   .8   .88  .7    .8   .176   .982 ,
          0  1     .87  .87   .7  1.053   .986 ,
          1   .9   .75  .68  1.3   .519   .98  ,
          0  1     .65  .65   .6   .519   .982 ,
          1   .95  .97  .92  1    1.23    .992 ,
          0   .95  .87  .83  1.9  1.354  1.02  ,
          0  1     .45  .45   .8   .322   .999 ,
          0   .95  .36  .34   .5  0      1.038 ,
          0   .85  .39  .33   .7   .279   .988 ,
          0   .7   .76  .53  1.2   .146   .982 ,
          0   .8   .46  .37   .4   .38   1.006 ,
          0   .2   .39  .08   .8   .114   .99  ,
          0  1     .9   .9   1.1  1.037   .99  ,
          1  1     .84  .84  1.9  2.064  1.02  ,
          0   .65  .42  .27   .5   .114  1.014 ,
```

```
0  1     .75  .75  1    1.322  1.004 ,
0   .5   .44  .22   .6   .114   .99  ,
1  1     .63  .63  1.1  1.072   .986 ,
0  1     .33  .33   .4   .176  1.01  ,
0   .9   .93  .84   .6  1.591  1.02  ,
1  1     .58  .58  1     .531  1.002 ,
0   .95  .32  .3   1.6   .886   .988 ,
1  1     .6   .6   1.7   .964   .99  ,
1  1     .69  .69   .9   .398   .986 ,
0  1     .73  .73   .7   .398   .986 ];
```

Column 1 is used for the binary response and columns 2, 5, and 7 are used for predictors. The design matrix X contains the intercept in its first column:

```
cnam = [ "remiss" "cell" "smear" "infil" "li" "blast" "temp" ];
remis = cname(remis,cnam);
nr = nrow(remis); y = remis[ ,1];
x = cons(nr,1,1.) -> remis[ ,[2 5 7]];
```

The following call of `glim` can be used to compute the parameter vector `par`:

```
model = " 1 = 2 5 7 ";
class = 1;
optn = [ "print"           3 ,
         "popt"            3 ,
         "pres"              ,
         "link"      "logit" ,
         "dist"   "binomial" ,
         "tech"     "trureg" ];
< gof,par,ase,cli,cov > = glim(remis,model,optn,class);
```

The following code could be used used to obtain the expected probabilities `yhc` of the logistic model. The conditional expression `yhd = yhc > .5;` computes the $(0, 1)$ vector of expected response values w.r.t. the cutoff value of .5:

```
par = [ -67.633906  -9.652152  -3.867100  82.07377 ]‘;
xp  = x * par;
yhc = 1. / (1. + exp(xp));
yhd = yhc .> .5;
```