CMAT Newsletter: March 2003

Wolfgang M. Hartmann

March 2003

1 General Remarks

1.1 Fixed Bugs

A number of bugs were fixed:

1. one factor specification for the `sem` function,
2. `global` statement in function definitions,
3. ...

1.2 Fitindices in `sem(.)`

The table of fit indices for the `sem(.)` function was reordered. For the ML fit of `sem12.inp` example we now report the following table:

```
(1) Standalone Fit Measures: -------------------------------
Fit criterion . . . . . .     0.3406
Chi-square (df = 8) . . . . . 16.6917 Prob>chi**2 = 0.0335
Elliptic cor. Chi-square. . . 11.9758 Prob>chi**2 = 0.1523
Normal Theory Reweighted LS Chi-square  . . . . .    22.1693
Probability of Close Fit  . . . . . .     0.0608
Z-Test of Wilson & Hilferty (1931). . . . . .     1.8336
(2) Incremental Fit Measures: ------------------------------
Null Model Chi-square (df = 15) . . . . . .      63.2479
RMSEA Estimate  . . . . .  0.1489  90%C.I.[ 0.0399,  0.2498]
ECVI Estimate . . . . . .  0.9597  90%C.I.[ 0.0000,  1.3088]
McDonald's (1989) Centrality. . . . . .     0.9168
Tucker-Lewis Coefficient TLI. . . . . .     0.6622
Bentler & Bonett's (1980) NFI . . . . . .     0.7361
Bentler's Comparative Fit Index . . . . . .     0.8199
Parsimonious NFI (James, Mulaik, & Brett,1982). .     0.3926
Bollen (1986) Normed Index Rho1 . . . . . .     0.5052
Bollen (1988) Non-normed Index Delta2 . . . . . .     0.8427
(3) Information Criteria: ----------------------------------
Akaike's Information Criterion. . . . . .     0.6917
Bozdogan's (1987) CAIC. . . . . .   -22.6045
Schwarz's Bayesian Criterion. . . . . .   -14.6045
(4) Other Fit Measures: ------------------------------------
```

```
Goodness of Fit Index (GFI) . . . . . .     0.9049
Parsimonious GFI (Mulaik, 1989) . . . . . .     0.4826
GFI Adjusted for Degrees of Freedom (AGFI). . . .     0.7503
Root Mean Square Residual (RMR) . . . . . .     0.1267
Hoelter's (1983) Critical N . . . . . .         47
```

1.3 Formatted Reading and Printing

In CMAT string data should normally be in apostrophes permitting unformatted reading. However, in practice formatted data sets occur which contain string data without apostrphes. The C-like functions `fopen`, `fscanf` and `fclose` could be used as demonstrated in:

1. `tsem\sem48.inp` for reading the `tdata\sem48.dat` data set
2. `test\tsample.inp` for reading the `tdata\diabetes.dat` and the `tdata\australian.dat` data set

Small test applications are shown in file `test\Tfile.inp`.
The data in `tdata\sem48.dat` start with the following 7 cases:

```
AMC       CONCORD             3224233224 A
AMC       PACER               3323435225 A
AMC       SPIRIT              3235333215 A
AUDI      4000                5443455444 E
AUDI      5000                4543553454 E
BUICK     CENTURY             3334444544 G
BUICK     ELECTRA             2434453555 G
```

For the format specification of these data we use:

1. starts with blank for newline white space
2. there is 1 blank white space before the last string
3. otherwise there is no white space.

The relevant statements in `tsem\sem48.inp` are

```
fid = fopen("..\\tdata\\sem48.dat","rt");
f = " %10c%20c%1i%1i%1i%1i%1i%1i%1i%1i%1i%1i%1i %1c";
d48 = fscanf(fid,f,.,13);
irc = fclose(fid); /* print d48; */
```

For reading the *Diabetes* and *Australian* data sets we use in `test\tsample.inp`:

```
form = fo = " %g";
for (j = 2; j <= 9; j++) form = strcat(form,fo);
fid = fopen("..\\tdata\\diabetes.dat","r");
data = fscanf(fid,form,.,9);

form = fo = " %g";
for (j = 2; j <= 15; j++) form = strcat(form,fo);
fid = fopen("..\\tdata\\australian.dat","r");
data = fscanf(fid,form,.,15);
```

The same data sets are used in `tsvm5.inp` and `tsvm6.inp`. In `tsvm3.inp` we also illustrate the use of the `fprintf(.)` function to write a 270×270 matrix:

```
form1 = "%20.17f %20.17f %20.17f %20.17f %20.17f\n";
form2 = strcat(form1,form1);
form  = form2;
for (j = 2; j <= 27; j++) form = strcat(form,form2);
/* print form; */
fid = fopen("..\\tdata\\heart.dat","w");
nbyt = fprintf(fid,form,rhes);
print "Bytes written:", nbyt;
fclose(fid);
```

1.4 New Functions

New functions are implemented:

cancor canonial correlations: At this time not all options are available which are in PROC CANCOR in SAS/STAT ([**?**]). Especially not the *canonical redundancy analysis*. My copy of Cooley & Lohnes (1971, [**?**]) was made by an old friend from Leipzig who was visiting Russia in the late seventies, and who obviously had not enough money for a complete copy. Therefore, the pages of the *redundancy analysis* are missing and I was not able to implement this method.

mburg modified Burg algorithms: This function implements the algorithm by Trindade (2003,[**?**]) in *JSS* in a very modified form:

1. The binary tree is not constructed in a recursive way; it is constructed layer by layer. This can save some memory.
2. The Hooke and Jeeves optimization algorithm was replaced by algorithms in the `nlp` function, especially the Nelder-Mead algorithm. This reduces the number of function calls by a factor from 50 until 1000.

sign2(a,b) returns the value y of the signum function:

$$y = \begin{cases} |a| & \text{if } b >= 0 \\ -|a| & \text{if } b < 0 \end{cases}$$

For scalar a and $b \neq 0$ there is $sign2(a, b) = |a|sign(b)$, but if a and a are objects of the same dimension, the `sign2` function is more general than `sign`.

sign4(a1,a2,a3,b) returns the value y of the signum function:

$$y = \begin{cases} a1 & \text{if } b < 0 \\ a2 & \text{if } b == 0 \\ a3 & \text{if } b > 0 \end{cases}$$

This version is the most general and permits missing values and string data in $a1$, $a2$, and $a3$. If b is $n \times m$ matrix, then y has the same dimension.

The illustration of this newsletter is an application of the `nlp(.)` function to a one-parameter IRT model.
Those of you who are interested in `sem(.)` I would like to remember that CMAT includes the `mardia(.)` function which computes p values for mv skewness and kurtosis as recently published by: Bonnett, D.G., Woodward, J.A., & Randall, R.L. (2002), "Estimating p-values for Mardia's coefficients of multivariate skewness and kurtosis", *Computational Statistics*, **17**, 117-122.

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 `cancor`

Purpose: This function is performing canonical correlation analysis.

Input: Two forms of model specification are permitted. The first two input arguments are the same for both forms, the last two arguments have a different meaning.

data specifies a $N \times n$ data matrix which contains ny columns for the response variables and the nx columns of predictor variables.

optn : The option argument is 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.

The first alternative for input arguments 3 and 4 is:

yind specifies an index vector containing the column numbers of response variables

xind specifies an index vector containing the column numbers of predictor variables

The second alternative for input arguments 3 and 4 is:

model : The analysis model is specified in form of a string, e.g. `model= "3=1 2"`, containing column numbers for variables. The model string specifies which variables (columns) are independend (predictors, covariates) and which are dependend (response variables) and is of the form

$$Y_1 \dots Y_r = X_{effect_1} \dots X_{effect_n}$$

where X_{effect_i} may be one of the following:

- a single variable x_i is an effect
- effects that are interactions among multiple x variables, seprated by the $*$ operator, $x_i * \dots * x_l$
- nested effects: that are single variables or interactions followed by a list of variables inside parentheses, $(x_i \dots x_l)$. Note that only categorical variables without interactions can be listed inside parentheses.

See `reg()` for more details.

class : 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.

The second alternative has the advantage that it permits nominally scaled predictor variables and the input of string data.

2.2 Function `mburg`

Purpose: This function implements AutoRegressive Moving Average (ARMA) models for one and two dimensional time series data.

Input: **x** Must be $N \times 1$ or $N \times 2$ data matrix for one resp. two dimensional time series data.

lags Must be k vector of positive lag values.

optn optional input of vector of options:

1. method: =1 or "yule": Yule-Walker
   =2 or "burg": Burg
   =3 or "morf": Vieira-Morf
   =4 or "nutt": Nuttall-Strand

2. integer specifying amount of printed output, =0 no printed output
3. =1: for mean corrected time series data, =0: for original values.
4. =1: optimize approximate likelihood for 2-dimensional data =0: compute exact optimal solution (for 2-dimensional data only for small number of lags possible), (valid only for two dimensional data, for onedimensional data always the exact solution is computed)
5. optimization technique (valid only for 2 dimensional data with `optn[4]=1`): = "nmsimp": Nelder-Mead algorithm (no derivatives) = "congra": conjugate gradient (only first order derivs) = "dbldog": double dogleg method (only first order derivs) = "duquan": dual quasi Newton method (only first order derivs) = "nrridg": Newton-Raphson ridge method (second order derivs) = "trureg": Trust Region method (second order derivs)

If `optn[4]=1` and `optn[5]` is not specified the Hooke and Jeeves optimization method is used as in Trindade (2003, [**?**]). This method can be very time consuming.

cov Optional input of the covariance matrix. Normally this is only known for simulated data.

2.3 Functions `sign2` and `sign4`

In addition to the already implemented `y = sign(b)` function two slightly more general forms are implemented:

sign2(a,b) returns the value y of the signum function like in Fortran. For integer and real argument b the result is

$$y = \begin{cases} |a| & \text{if } b >= 0 \\ -|a| & \text{if } b < 0 \end{cases}$$

For scalar a and $b \neq 0$ there is $sign2(a, b) = |a|sign(b)$, but if a and a are objects of the same dimension, the `sign2` function is more general than `sign`.

sign4(a1,a2,a3,b) returns the value y of a more general signum function.

For integer and real argument b the result is

$$y = \begin{cases} a1 & \text{if } b < 0 \\ a2 & \text{if } b == 0 \\ a3 & \text{if } b > 0 \end{cases}$$

This version is the most general and permits missing values and string data in $a1$, $a2$, and $a3$. If b is $n \times m$ matrix, then y has the same dimension.

3 Illustration: ML Estimation of Item Response Model

This example simulates data from a one-parameter (Rasch) item response model and then estimates the parameters by maximum likelihood.
The data represent *nsub* subjects who take a test containing *nitem* items. Each subject has an ability parameter, and each item has a difficulty parameter. The probability of a correct answer to an item is a logistic function of the difference between the subject's ability and the item's difficulty. See, for example, Hambleton, Swaminathan & Rogers (1991).

The probability of a correct response for subject n on item i is

$$p_{ni} = \frac{\exp(a_n - d_i)}{1 + \exp(a_n - d_i)} \tag{1}$$

where

$$\begin{aligned} a_n &= \text{ability of subject n,} \\ d_i &= \text{difficulty of item i.} \end{aligned}$$

If the responses are coded:

$$x_{ni} = \begin{cases} 1 & \text{for a correct resonse} \\ 0 & \text{for an incorrect resonse} \end{cases}$$

then the log likelihood is:

$$\max l(a_n, d_i) = \sum_n \sum_i x_{ni} \log(p_{ni}) + (1 - x_{ni}) \log(1 - p_{ni}) \tag{2}$$

which is to be maximized with respect to the *nsub* subject abilities a_n and the *nitem* item difficulties d_i. The first derivatives of the probabilities p_{ni} with respect to the subject abilities a_n are,

$$\frac{\partial p_{ni}}{\partial a_n} = \frac{\exp(a_n - d_i)}{(1 + \exp(a_n - d_i))^2} \tag{3}$$

and the first derivatives of the probabilities p_{ni} with respect to the item difficulties d_i are,

$$\frac{\partial p_{ni}}{\partial d_i} = -\frac{\exp(a_n - d_i)}{(1 + \exp(a_n - d_i))^2} \tag{4}$$

The following code generates a 30×20 data set with $nitem = 20$ item variables and $nsub = 30$ observations.

```
srand(123);
nsub = 30; nitem = 20;
n = nsub + nitem - 1;

/* (A) Generate Data: exam2[nsub.nitem] in [0,1] */
exam2 = cons(nsub,nitem);
item = cons(nitem);
kd = .5 * (1. + nitem);
diff = [ 1. : nitem ];
diff = 4. *(diff - kd) / (real)nitem;
ka = .5 * (1. + nsub);
abil = [ 1. : nsub ];
abil = 4. * (abil - ka) / (real)nsub;
for (isub = 1; isub <= nsub; isub++) {
  ab = abil[isub]; rd = rand(nitem);
  for (item = 1; item <= nitem; item++) {
    x = exp(ab - diff[item]);
    p = x / (1. + x); d = p - rd[item];
    exam2[isub,item] = .5*(sign(d) + 1.);
} }
print "Data Matrix", exam2;
```

The maximum-likelihood estimates of the $nsub = 30$ ability parameters and $nitem = 19$ difficulty parameters are computed by maximizing a logistic function with different optimization techniques. We will compare three second order and three first order optimization techniques:

1. second order: *Newton-Raphson ridge* technique `"nrridg"`
2. second order: *trust-region* technique `"trureg"`
3. second order: *Newton-Raphson line search* technique `"newrap"`
4. first order: *quasi Newton* technique `"duquan"`
5. first order: *double dogleg* technique `"dbldog"`
6. first order: *conjugate gradient* technique `"congra"`

We decided to compare four approaches of specifying function and derivatives:

1. no Hessian needed: (1.1) scalar and (1.2) matrix notation: we also present results of second order techniques which use finite difference approximation of Hessian
2. Hessian is used: (2.1) scalar and (2.2) matrix notation: we also present results of first order techniques where the former approaches would be more appropriate

3.1 (1.1) only gradient is needed: scalar notation

```
/*--- (1) If only gradient is needed: use (agrd,dgrd) Globals ---*/
/*--- (1.1) Scalar Version ---*/
/* Define global vectors: gradients */
/* Define global vectors: gradients */
agrd = cons(1,nsub); dgrd = cons(1,nitem);

/* function evaluation */
function f11_irt(x) global(exam2,agrd,dgrd) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  agrd[1:nsub] = 0.; dgrd[1:nitem] = 0.;
  npar = ncol(x); ns1 = nsub + 1;
  abil = x[1:nsub]; diff = [ 0. ] -> x[ns1:npar];
  loglik = 0.;
  for (isub = 1; isub <= nsub; isub++) {
    xitm = exam2[isub,]; ab = abil[isub];
    for (item = 1; item <= nitem; item++) {
      t = exp(ab - diff[item]);
      tt = 1. + t; tt = t / (tt * tt);
      r = t / (1. + t); dpda = tt;
      if (xitm[item] == 0) { r = 1. - r; dpda = -dpda; }
      loglik += log(r); agrd[isub] += dpda / r;
      if (item > 1) dgrd[item] -= dpda / r;
  } }
  return(loglik);
}
```

```
/* gradient evaluation */
function g1_irt(x) global(exam2,agrd,dgrd) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  grad = agrd -> dgrd[2:nitem];
  return(grad);
}

x0 = cons(1,n,1.);
crit = f11_irt(x0);
print "CRIT=", crit;
grad = g1_irt(x0);
print "grad=", grad;
```

Inappropriate method: Hessian must be computed by finite differences:

```
x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "nrridg" ];
< xr, rp > = nlp(f11_irt,x0,mopt,.,g1_irt);
```

Inappropriate method: Hessian must be computed by finite differences:

```
x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "trureg" ];
< xr, rp > = nlp(f11_irt,x0,mopt,.,g1_irt);
```

Inappropriate method: Hessian must be computed by finite differences:

```
x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "newrap" ];
< xr, rp > = nlp(f11_irt,x0,mopt,.,g1_irt);
```

Appropriate method: Hessian is not needed:

```
x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "duquan" ];
< xr, rp > = nlp(f11_irt,x0,mopt,.,g1_irt);
```

Appropriate method: Hessian is not needed:

```
x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "dbldog" ];
< xr, rp > = nlp(f11_irt,x0,mopt,.,g1_irt);
```

Appropriate method: Hessian is not needed:

```
x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "congra" ];
< xr, rp > = nlp(f11_irt,x0,mopt,.,g1_irt);
```

3.2 (1.2) only gradient is needed: matrix notation

```
/*--- (1.2) Matrix Version: without do loops ---*/
/* Define global vectors: gradients */
agrd = cons(1,nsub); dgrd = cons(1,nitem);

function f12_irt(x) global(exam2,agrd,dgrd) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem];
            exam2[nsub,nitem], agrd[nsub], dgrd[nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  agrd[1:nsub] = 0.; dgrd[1:nitem] = 0.;
  npar = ncol(x); ns1 = nsub + 1;
  abil = x[1:nsub]; diff = [ 0. ] -> x[ns1:npar];

  /* hm1[nsub,nitem] matrix with abil in cols */
  hm1 = abil‘ @ cons(1,nitem,1.);
  /* hm2[nsub,nitem] matrix with diff in rows */
  hm2 = diff  @ cons(nsub,1,1.);
  dd = exp(hm1 - hm2);
  d1 = 1. + dd; r1 = dd ./ d1;

  /* generate (-1,1) matrix for subtraction:
     two large [nsub,nitem] matrices: */
  hm1 = exam2 - .5;
  hm1 = sign(hm1); hm2 = .5*(1. - hm1);
  /* r1 = 1. - 1. * r1 or r1 = r1 */
  r1 = hm2 + hm1 .* r1;
  r2 = log(r1);
  loglik = r2[+,+];
  d1 = d1 .* d1; r2 = dd ./ d1;
  r2 = hm1 .* r2 ./ r1;
  agrd = r2[,+]‘; dgrd = -r2[+,];
  return(loglik);
}

/* gradient evaluation */
function g1_irt(x) global(exam2,agrd,dgrd) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  grad = agrd -> dgrd[2:nitem];
  return(grad);
}
```

The same optimization techniques are applied as above in (1.1).

3.3 (2.1) Hessian is specified: scalar notation

```
/*--- (2.1) Scalar Version ---*/
/* Define global matrices: for grad[] and hess[] */
xout = cons(nsub,nitem); pout = cons(nsub,nitem);
```

```
/* function evaluation */
function f21_irt(x) global(exam2,xout,pout) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  npar = ncol(x); ns1 = nsub + 1;
  abil = x[1:nsub]; diff = [ 0. ] -> x[ns1:npar];

  loglik = 0.;
  for (isub = 1; isub <= nsub; isub++) {
    xitm = exam2[isub,]; ab = abil[isub];
    for (item = 1; item <= nitem; item++) {
      t = exp(ab - diff[item]);
      r = t / (1. + t);
      if (xitm[item] == 0) r = 1. - r;
      xout[isub,item] = t; pout[isub,item] = r;
      loglik += log(r);
  } }
  return(loglik);
}

/* gradient evaluation */
function g21_irt(x) global(exam2,xout,pout) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  npar = ncol(x); ns1 = nsub + 1;
  agrd = cons(1,nsub); dgrd = cons(1,nitem);
  abil = x[1:nsub]; diff = [ 0. ] -> x[ns1:npar];
  grad = cons(npar,1);

  for (isub = 1; isub <= nsub; isub++) {
    xitm = exam2[isub,]; ab = abil[isub];
    for (item = 1; item <= nitem; item++) {
      t = xout[isub,item]; r = pout[isub,item];
      tt = 1. + t; tt = t / (tt * tt);
      dpda = tt; dpdd = -tt;
      if (xitm[item] == 0) { dpda = -dpda; dpdd = -dpdd; }
      agrd[isub] += dpda / r;
      if (item > 1) dgrd[item] += dpdd / r;
  } }
  grad = agrd -> dgrd[2:nitem];
  return(grad);
}
```

To avoid time-consuming symmetry swapping, first only the lower triangular matrix is computed and then defined as symmetric by the `(tri2sym)` casting.

```
/* Hessian: scalar notation */
function h21_irt(x) global(exam2,xout,pout) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  npar = ncol(x); ns1 = nsub + 1;
```

```
  abil = x[1:nsub]; diff = [ 0. ] -> x[ns1:npar];
  hess = cons(npar,npar);

  for (isub = 1; isub <= nsub; isub++) {
    xitm = exam2[isub,]; ab = abil[isub];

    /* diagonal entry [isub,isub] */
    /* row and column isub: hess[nsub+ij,isub] */
    /* diagonal [nsub+ij,nsub+ij], ij=1,...,nitem-1 */
    s = 0.; ij = nsub;
    for (item = 1; item <= nitem; item++, ij++) {
      t = xout[isub,item]; r = pout[isub,item];
      r1 = 1. + t; r2 = t / (r1 * r1);
      d1 = r2 - 2. * r2 * r2 * r1;
      if (xitm[item] == 0) d1 = -d1;
      d2 = (d1 * r - r2 * r2) / (r * r);
      s += d2;
      /* now d1 has the other sign */
      if (item > 1) {
        hess[ij,ij] += d2;
        hess[isub,ij] = -d2;
    } }
    hess[isub,isub] = s;
  }
  hess = (tri2sym)hess;
  return(hess);
}

x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "nrridg" ];
< xr, rp > = nlp(f21_irt,x0,mopt,.,g21_irt,h21_irt);

x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "trureg" ];
< xr, rp > = nlp(f21_irt,x0,mopt,.,g21_irt,h21_irt);

x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "newrap" ];
< xr, rp > = nlp(f21_irt,x0,mopt,.,g21_irt,h21_irt);

x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "duquan" ];
< xr, rp > = nlp(f21_irt,x0,mopt,.,g21_irt);

x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "dbldog" ];
< xr, rp > = nlp(f21_irt,x0,mopt,.,g21_irt);
```

```
x0 = cons(1,n,1.);
mopt = [ "max"         ,
         "tech" "congra" ];
< xr, rp > = nlp(f21_irt,x0,mopt,.,g21_irt);
```

3.4 (2.2) Hessian is specified: matrix notation

```
/*--- (2.2) Matrix Version: without do loops ---*/

/* Define global matrices: for grad[] and hess[] */
xout = cons(nsub,nitem);
pout = cons(nsub,nitem);

/* function evaluation */
function f22_irt(x) global(exam2,xout,pout) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  npar = ncol(x); ns1 = nsub + 1;
  abil = x[1:nsub]; diff = [ 0. ] -> x[ns1:npar];

  /* hm1[nsub,nitem] matrix with abil in cols */
  hm1 = abil‘ @ cons(1,nitem,1.);
  /* hm2[nsub,nitem] matrix with diff in rows */
  hm2 = diff  @ cons(nsub,1,1.);
  xout = exp(hm1 - hm2);
  d1 = 1. + xout; r1 = xout ./ d1;

  /* pout = 1. - 1. * r1 or pout = r1 */
  /* generate (-1,1) matrix for subtraction:
     two large [nsub,nitem] matrices: */
  hm1 = exam2 - .5;
  hm1 = sign(hm1); hm2 = .5*(1. - hm1);
  pout = hm2 + hm1 .* r1;
  r2 = log(pout);
  loglik = r2[+,+];
  return(loglik);
}

/* gradient evaluation */
function g22_irt(x) global(exam2,xout,pout) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  agrd = cons(nsub); dgrd = cons(nitem);

  hm1 = exam2 - .5; hm1 = sign(hm1);
  tt = 1. + xout; tt = xout ./ (tt .* tt);
  dpda = (hm1 .* tt) ./ pout;
  agrd = dpda[,+]‘; dgrd = -dpda[+,];
  grad = agrd -> dgrd[2:nitem];
  return(grad);
}
```

To avoid time-consuming symmetry swapping, first only the lower triangular matrix is computed and then defined as symmetric by the `(tri2sym)` casting.

```
/* Hessian */
function h22_irt(x) global(exam2,xout,pout) {
  /* parms: x[nsub+nitem-1] = abil[1:nsub] diff[2:nitem] */
  nsub = nrow(exam2); nitem = ncol(exam2);
  npar = ncol(x); ns1 = nsub + 1;
  abil = x[1:nsub]; diff = [ 0. ] -> x[ns1:npar];

  hess = cons(npar,npar);
  n1 = nsub+1; n2 = nsub+nitem-1; ind = [n1:n2];

  for (isub = 1; isub <= nsub; isub++) {
    hm = exam2[isub,] - .5; hm = sign(hm);
    ab = abil[isub];

    /* diagonal entry [isub,isub] */
    tm = xout[isub,]; rm = pout[isub,];
    r1 = 1. + tm; r2 = tm ./ (r1 .* r1);
    dm1 = r2 - 2. .* r2 .* r2 .* r1;
    dm1 = hm .* dm1;
    dm2 = (dm1 .* rm - r2 .* r2) ./ (rm .* rm);
    dm1 = dm2[2:nitem];
    hess[isub,ind] = -dm1;
    hess[isub,isub] = dm2[+];
    j = n1;
    for (i = 2; i <= nitem; i++, j++) hess[j,j] += dm2[i];
  }
  hess = (tri2sym)hess;
  return(hess);
}
```

The same optimization techniques are applied as above in (2.1).

3.5 Comparison of Computer Time

The following table shows the computer time in seconds, the number of iterations, function calls, and gradient or Hessian calls that are needed by various optimization techniques to obtain ML estimates with about the same precision:

Spec	NRRIDG	TRUREG	NEWRAP	DUQUAN	DBLDOG	CONGRA
(1.1)	137 : 7,4,7	159 : 9,4,8	182 : 7,9,7	9 : 22,17,0	12 : 19,17,0	17 : 39,27,0
(1.2)	29 : 7,4,7	34 : 9,4,8	39 : 7,9,7	2 : 22,17,0	3 : 19,17,0	4 : 39,27,0
(2.1)	19 : 7,4,7	23 : 9,4,8	39 : 7,9,7	25 : 22,17,0	32 : 19,17,0	44 : 39,27,0
(2.2)	10 : 7,4,7	12 : 9,4,8	16 : 7,9,7	2 : 22,17,0	3 : 19,17,0	3 : 39,27,0

Those techniques that use second-order derivatives (TRUREG, NRRIDG, and NEWRAP) do not need as many function and gradient calls; however, they need more computation time for

- computing the Hessian matrix either by finite difference approximation or executing the program statements for the analytic specification,

- solving the linear system for the search direction.

The size of the problem and the expense of computing the Hessian matrix decide which optimization technique is most suitable for a special application. For this type of application using first order techniques seem to be the more economic choice.