C++ Functions in Maxliklib Library

Shelby J. Haberman
Haberman Statistics

Abstract

The functions in the maxliklib repository are described. Arguments and their definitions are specified, and dependencies of functions are stated.

Keywords: Maximization procedures, quadrature procedures, maximum likelihood

The maxliklib repository consists of C++ functions helpful in estimation related to maximum likelihood. The functions should be appropriate for C++14. They rely on the Armadillo library (Sanderson & Curtin, 2016, 2018) at http://arma.sourceforge.net, the StatsLib library at https://www.kthohr.com/statslib.html, and the Boost library at https:/www.boost.org. Unless otherwise noted, for the library members considered, it is assumed that users have verified that function arguments are valid. Namespaces assumed where relevant are *std*, *arma*, and *boost::math*. The following functions are found in the library, with files in the source code with the suffix .cpp.

- addsel
- berresp
- betad
- conjgrad
- contresp
- cumresp
- dirichlet
- dupname

Shelby J. Haberman https://orcid.org/0000-0002-5490-0405
Shelby Haberman is an independent statistical consultant whose website is https://www.habermanstatistics.com. He can also be reached at haberman.statistics@gmail.com.

- eaps
- fitquad
- gammad
- genfact
- genprods
- genresp
- genresplik
- genrespmle
- gpcm
- gradascent
- gradresp
- gumbell
- gumbelu
- hermcoeff
- hermpoly
- hermpw
- intsel
- invcdf
- irtm
- irtmle
- irtms
- irtmsave
- irtmsaves
- ivecsel
- lcumresp

- ldscore
- lgradresp
- linsel
- lmultlogit
- logistic
- logit
- loglogl
- loglogu
- logmean
- lscore
- lw
- lwm
- maxberresp
- maxf2vvar
- maxlinq2
- maxquad
- maxselect
- multlogit
- modit
- nanf2v
- normal
- nrv
- oneparamirt
- pcm
- posterior

- posteriors
- probit
- qnormpwe
- ranklogit
- rebound
- regprod
- regprodf
- savmaxf2v
- sintsel
- sivecsel
- startoneparamirt
- startgpcm
- startpcm
- starttwoparamirt
- svecsel
- truncresp
- twoparamirt
- vecsel
- wmc

Distributions of Sums of Independent Multinomial Variables

The functions in this section implement a modified and generalized version of the Lord-Wingersky algorithm (Lord & Wingersky, 1984; Thissen et al., 1995). The numerical procedures and their rationale are discussed in lw.pdf.

lw

The function *lw* finds the probability mass function of the sum S of mutually independent Bernoulli random variables X_j, $0 \leq j < n$. The function declaration is

vec lw(const double & c, const vec & p).

In the function definition, c is a constant c, and p is an n-dimensional vector $\mathbf{p}$. The vector $\mathbf{p}$ with elements p_j, $0 < p_j < 1$, has dimension n. For $0 \leq j < n$, p_j is the probability that $X_j = 1$. The variable c is normally a small positive number used as in lw.pdf to remove very small probabilities from consideration in order to speed computation. If c is not positive, then the modified Lord-Wingersky algorithm used by *lw* reduces to the conventional algorithm. The probability mass function is provided by *lw*, a vector with $n + 1$ elements. For $0 \leq k \leq n$, element k of *lw* is the probability $P(S = k)$ that $S = k$.

lwm

The function *lwm* finds the probability mass function of the sum S of n mutually independent random variables X_j, $0 \leq j < n$ with integer values from 0 to $I_j - 1$ for an integer $I_j > 1$. The function declaration is

vec lwm(const double & c, const field<vec> & p).

Here p has n vector members $\mathbf{p}_j$, $0 \leq j < n$. For each j, $\mathbf{p}_j$ has positive elements p_{ij}, $0 \leq i < I_j$, that sum to 1. The probability $P(X_j = i)$ is p_{ij}. The vector *lwm* has $K + 1$ elements, where $K = \sum_{j=1}^{n}(I_j - 1)$ and $P(S = k)$ is element k of *lwm* for $0 \leq k < K$. The definition of c is the same as in lw.

Tools for Line Searches

The functions in this section facilitate line searches during function maximization. Throughout discussions in this section and in Functions related to the Newton-Raphson algorithm and Functions Related to Gradient Methods, the theoretical background and the definitions of η, γ_1, γ_2, and κ are found in convergence.pdf. For some positive integer p and nonempty open convex set O of p-dimensional vectors, a continuously differentiable real function f on O is to be maximized by an iterative algorithm with a starting value in O. It is assumed that, for some real a, the set A of members of O at which f is at least a is closed and bounded, and the set A_0 of members of O at which f exceeds a is nonempty. The function f is assumed to be strictly pseudoconcave on A_0. The starting values for algorithms are assumed to be in A_0. The convention is adopted that $f(\mathbf{x}) = -\infty$ if $\mathbf{x}$ is not in O.

maxlinq2

The function *maxlinq2* provides a line search for maximization algorithms. Only function values and gradients are used when *order* is 1, but Hessian matrices or their approximations are computed if *order* is greater than 1. The function declaration is

maxf2v maxlinq2(const int & order, const params & mparams, const vec & v,
const maxf2v & vary0, const function<f2v(const int &, const vec &)>f).

Here the definition of *maxf2v* is

struct maxf2v{vec locmax; double max; vec grad, mat hess;};,

For a vector $\mathbf{y}$, the corresponding definitions of *maxf2v* are *maxf2v.locmax* equals $\mathbf{y}$, *maxf2v.max* equals $f(\mathbf{y})$, *maxf2v.grad* equals the gradient $\nabla f(\mathbf{y})$ of f at $\mathbf{y}$, and, for *order* greater than 1, *maxf2v.hess* equals the Hessian matrix $\nabla^2 f(\mathbf{y})$ of f at $\mathbf{y}$. The corresponding definition of *f2v* is

struct f2v{double value; vec grad; vec hess};,

where, for the vector $\mathbf{y}$, *f2v.value* is $f(\mathbf{y})$, *f2v.grad* is $\nabla f(\mathbf{y})$, and, for *order* greater than 1, *f.hess* is $\nabla^2 f(\mathbf{y})$. In the arguments of *maxlinq2*, *vary0.locmax* is the starting vector $\mathbf{x}$ for the line search, the search direction is $\mathbf{v}$, and *f* is the function f together with its gradient and, fot *order* greater than 1, its Hessian matrix. Maximization details are defined by *mparams*, where the definition of *params* is

struct params{bool print; int maxit; int maxits; double eta;
double gamma1; double gamma2; double kappa; double tol;}.

Here *mparams.print* is used for output of the iteration number and function value at the end of the iteration, *mparams.maxit* is the number of primary iterations, *mparams.maxits* is the maximum number of uses of maxquad permitted for each primary iteration, *mparams.eta* is η, *mparams.gamma1* is γ_1, *mparams.gamma2* is γ_2, and *mparams.kappa* is κ. Iterations cease if the function value changes less than *mparams.tol* after a primary iteration.

In *maxlinq2*, *maxlinq2.locmax* is an approximate location $y = \mathbf{x} + \alpha\mathbf{v}$ of the maximum over α of $f(\mathbf{x} + \alpha\mathbf{v})$. The functions maxf2vvar, maxquad, modit, and rebound are all used.

maxquad

The function *maxquad* finds the location x of the maximum of a quadratic function q on the real line with value $q(x_0)$ and derivative $q_1(x_0)$ at x_0 and value $q(x_1)$ at $x_1 \neq x_0$. It is assumed that $q(x_0) \geq q(x_1)$. For a given $s > 0$, the function returns x unless $|x - x_0| > s$. If $|x - x_0| > s$ and $q_1(x_0) > 0$, then $x_0 + s$ is returned. If $|x - x_0| > s$ and $q_1(x_0) < 0$, then $x_0 - s$ is returned. The function declaration is

double maxquad(const double & x0, const double & x1, const double & f0, const double & f1, const double & g0, const double & stepmax).

Here *x0* is x_0, *x1* is x_1 used, *f0* is $q(x_0)$, *f1* is $q(x_1)$, *g0* is $q_1(x_0)$, and *stepmax* is s.

modit

The function *modit* truncates an iteration to conform to limits on step size and bounds in the case of a real function of one variable with a unique critical point and a limit of $-\infty$ as the absolute value of the function argument approaches ∞. The function declaration is

double modit(const double & eta, const double & alpha0, const double & alpha1, const double & stepmax, const bounds & b),

and the struct *bounds* is defined as

struct bounds {double lower; double upper;}.

Here *eta* corresponds to η, *alpha0* is the previous location α_0, *alpha1* is the proposed new location α_1, *stepmax* is the positive limit s on step size, *b.lower* is the lower bound $b_L \geq -\infty$, and *b.upper* is the upper bound $b_U \leq \infty$. It is assumed that $\alpha_0 \neq \alpha_1$ and $b_L \leq \alpha_0 \leq b_U$. If $\alpha_1 > \alpha_0$, then $b_U > \alpha_0$ and the function returns the minimum of α_1, $\alpha_0 + s$, and $\alpha_0 + \eta(b_U - \alpha_0)$. If $\alpha_1 < \alpha_0$, then $b_L < \alpha_0$ and the function returns the maximum of α_1, $\alpha_0 - s$, and $\alpha_0 + \eta(b_L - \alpha_0)$.

rebound

The function *rebound* updates the lower and upper bounds for maximization of a differentiable real function f on the real line with a unique critical point and a limit of $-\infty$ as the absolute value of the function argument approaches ∞. The function declaration is

bounds rebound(const double & y, const double & der, const bounds & b).

The struct *bounds* is defined as in modit. Here y is the current location y, *der* is the function derivative $f_1(y)$ at y, *b.lower* is the current lower bound L, and *b.upper* is the current upper bound U. It is assumed that $f_1(y) \neq 0$. If $f_1(y) > 0$, the returned lower bound is y and the returned upper bound is U. If $f_1(y) < 0$, the returned lower bound is L and the returned upper bound is y.

Functions related to the Newton-Raphson algorithm

In this section, functions are discussed that are related to the Newton-Raphson algorithm. It should be noted that references to function values, gradients, and Hessian matrices do not address computational methods. In fact, the function values, gradients, and Hessian matrices employed may be approximations derived by numerical differentiation or large-sample approximations. In this section, f is assumed to be twice continuously differentiable.

maxf2vvar

The function *maxf2vvar* is used to combine information on a location and on a function's value, gradient, and Hessian matrix at the location. The function *maxf2vvar* has declaration

maxf2v maxf2vvar(const int & order, const vec & y, const f2v & fy);.

The structs *f2v* and *maxf2v* are defined as in maxlinq2. At the vector $\mathbf{y}$, *fy.value* is $f(\mathbf{y})$, *fy.grad* is $\nabla f(\mathbf{y})$ if *order* is positive, and *fy.hess* is $\nabla^2 f(\mathbf{y})$ if *order* is greater than 1. The returned value *maxf2vvar.locmax* is $\mathbf{y}$, while *maxf2vvar.max* is $f(\mathbf{y})$, *maxf2var.grad* is $\nabla f(\mathbf{y})$ if *order* is positive, and *maxf2var.hess* is $\nabla^2 f(\mathbf{y})$ if *order* is greater than 1.

nrv

The function *nrv* applies a modified version of the Newton-Raphson algorithm to maximization of the function f. The function *nrv* has declaration

maxf2v nrv(const int & order, const params & mparams, const vec & start,
const function<f2v(const int &, vec &)> f).

The structs *f2v*, *maxf2v*, and *params* are defined as in maxlinq2. The algorithm is found in convergence.pdf. The starting vector $\mathbf{y}_0$ must be in O. The function returns the approximation $\mathbf{y}$ for the location of the maximum, together with $f(\mathbf{y})$, $\nabla f(\mathbf{y})$ and $\nabla^2 f(\mathbf{y})$.

The function *nrv* uses maxf2vvar, maxlinq2, maxquad, modit, and rebound. The value of *order* should be at least 2. The starting vector is *start*, and the function, gradient, and Hessian matrix are provided in *f*.

savmaxf2v

The function *savmaxf2v* is used to save basic results of maximization. The declaration is

void savmaxf2v(const int & order , const maxf2v & vlm, const string & out,
const bool & fflag, const bool & pflag).

The value of *order* is 1, 2, or 3. The struct *maxf2v* is defined as in maxlinq2. If *fflag* is *true*, then the output file is defined by *out*. The output file must be a binary file used by Armadillo for storage. If *pflag* is *true*, then standard output is employed. Both *fflag* and *pflag* may have the same value. If *order* is 1, output is *vlm.max*, *vlm.locmax*, and *vlm.grad*. If *order* exceeds2, then output also includes *vlm.hess*, the inverse of -1 times *vlm.hess*, and the square roots of the diagonal elements of this inverse.

Functions Related to Gradient Methods

In this section, functions are considered based on gradient-based methods.

conjgrad

The function *conjgrad* implements a conjugate gradient algorithm for maximization of the function f. The function declaration is

maxf2v conjgrad(const int & order, const params & mparams,
const vec & start, const function<f2v(const int & , const vec &)> f).

Arguments are defined as *nrv*. The value of *order* must be at least 1. If *order* is at least 2, Hessian matrices are computed even though not used in the algorithm.
The function *conjgrad* uses maxf2vvar, maxlinq2, maxquad, modit, and rebound.

gradascent

The function *gradascent* uses a gradient-ascent algorithm for maximization of f. The function declaration for *gradascent* is

maxf2v gradascent(const order & , const params & mparams,
const vec & start, const function<f2v(const int & , const vec &)> f).

The functions maxf2vvar, maxlinq2, maxquad, modit, and rebound are used. Definitions are as in conjgrad.

maxselect

The function *maxselect* uses either the conjugate-gradient, gradient-ascent, or modified Newton-Raphson algorithm for maximization of f. The function declaration for *maxselect* is

maxf2v maxselect(const order & , const params & mparams, const char & algorithm const vec & start, const function<f2v(const int & , const vec &)> f).

The functions conjgrad, gradascent, maxf2vvar, maxlinq2, maxquad, modit, nrv, and rebound are used. Definitions are as in conjgrad, except that *algorithm* is C for conjugate gradient, G for gradient ascent, and N for Newton-Raphson.

Log-likelihood Components

In this section, components of log-likelihood functions are provided. A component has the form $\ell_c(\boldsymbol{\beta}; \mathbf{Y}, A, F, q, r)$ and corresponds to a model for a single response variable $\mathbf{Y}$. Here the character A defines the type of model component involved, F is a distribution function with a positive and twice-continuously differential derivative F_1 such that $\log F_1$ has a negative second derivative. The integer $q > 0$ is the parameter dimension, and the integer $r > 0$ is the data dimension. The character A is in the set $\mathcal{A}$ with elements B (logit beta), C (cumulative), c (cumulative with slope parameter), D (continuous), E (logit Dirichlet), G (graded), g (graded with slope parameter), H (log gamma), L (multinomial logit), l (multinomial logit with slope parameter), M (maximum of two independent Bernoulli variables), N (multivariate normal), P (log-mean Poisson case), R (rank logit), S (Bernoulli), and T (censored continuous). Distribution functions used in this section are in the set $\mathcal{F}$ with four members, G_L, the standard minimum Gumbel distribution function with value $G_L(y) = 1 - \exp(-\exp(y))$ for y real (Gumbel, 1935), G_U, the standard maximum Gumbel distribution with value $G_U(y) = \exp(-\exp(-y))$ for y real, Ψ, the standard logistic distribution function with value $\Psi(y) = 1/[1+\exp(-y)]$ for y real, and Φ, the standard normal distribution function with derivative $\Phi_1(y) = \exp(-y^2/2)/(2\pi)^{1/2}$ for real y. The value of F is only relevant in the cumulative, continuous, graded, Bernoulli, and censored continuous cases. The variables M, F, q, and r then define an open convex subset $O(A, F, q, r)$ of q-dimensional vectors, a set $\mathcal{Y}(A, F, q, r)$ of distributions on a set $D(A, F, q, r)$ of r-dimensional vectors, and a function $\zeta(\cdot; A, F, q, r)$ from $O(A, F, q, r)$ onto $\mathcal{Y}(A, F, q, r)$ with value $\zeta(\boldsymbol{\beta}; A, F, q, r)$ at $\boldsymbol{\beta}$. The component corresponds to the model that $\mathbf{Y}$ has distribution $\zeta(\boldsymbol{\beta}; A, F, q, r)$ for some $\boldsymbol{\beta}$ in $O(A, F, q, r)$.

Three cases are considered for $D(A, F, q, r)$ and $\boldsymbol{\beta}$ in $O(A, F, q, r)$:

1. The set $D(A, F, q, r)$ is convex and has a nonempty interior, and $\zeta(\boldsymbol{\beta}; A, F, q)$ has a positive and continuous density function $f(\cdot; \boldsymbol{\beta}, A, F, q, r)$.

2. The set $D(A, F, q, r)$ is finite or countably infinite set, and $\zeta(\boldsymbol{\beta}; A.F, q, r)$ has a positive probability mass function $f(\cdot; \boldsymbol{\beta}, A, F, q, r)$ on $D(A, F, q, r)$.

3. The set $D(A, F, q, r)$ is the union of the nonempty disjoint sets $D_c(A, F, q, r)$ and $D_d(A, F, q, r)$, $D_c(A, F, q, r)$ is convex and has a nonempty interior, $D_d(A, F, q, r)$ is finite or countably infinite, the restriction of $\zeta(\boldsymbol{\beta}; A, F, q, r)$ to $D_c(A, F, q, r)$ has a positive and continuous density function $f_c(\cdot; \boldsymbol{\beta}, A, F, q, r)$, the restriction of $\zeta(\boldsymbol{\beta}; A, F, q, r)$ to $D_d(A, F, q, r)$ has a positive mass function $f_d(\cdot; \boldsymbol{\beta}, A, F, q, r)$, and $f(\mathbf{y}; \boldsymbol{\beta}; A, F, q, r)$ is $f_c(\mathbf{y}; \boldsymbol{\beta}; A, F, q, r)$ for $\mathbf{y}$ in $D_d(A, F, q, r)$ and $f_d(\mathbf{y}; \boldsymbol{\beta}, S, F, q, r)$ for $\mathbf{y}$ in $D_d(A, F, q, r)$.

In all cases, the log-likelihood component $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ is $\log(f(\mathbf{y}; \boldsymbol{\beta}, A, F, q, r))$ for $\mathbf{y}$ in $D(A, F, q, r)$. It is assumed that $\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ is twice continuously differentiable with gradient function $\nabla\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ and corresponding Hessian matrix $\nabla^2\ell_c(\cdot; \mathbf{y}, A, F, q, r)$.

The log-likelihood components are combined to form a weighted log-likelihood based on $n \geq 1$ responses. For response i, $0 \leq i < n$, the corresponding model is defined by positive integers q_i and r_i and character variables A_i in $\mathcal{A}$ and F_i in $\mathcal{F}$. The component of the log likelihood for observation i involves the predicted random vector $\mathbf{Y}_i$, the q_i by p predicting matrix $\mathbf{X}_i$ in a nonempty set $\mathcal{X}_i$, the q_i-dimensional vector $\mathbf{o}_i$, and the positive real weight w_i. Assume that $\mathbf{X}_0$ is fixed. For any response $i > 0$, assume that $\mathbf{X}_i$ is fixed or is a function of the $\mathbf{Y}_j$, $j < i$, and assume that the conditonal distribution of $\mathbf{Y}_i$ given $\mathbf{Y}_j$, $j < i$, is the same as the conditional distribution of $\mathbf{Y}_i$ given $\mathbf{X}_i$. Let $\boldsymbol{\lambda}_i(\boldsymbol{\beta}) = \mathbf{o}_i + \mathbf{X}_i\boldsymbol{\beta}$ be in $O(A_i, F_i, q_i, r_i)$ for $0 \leq i < n$. Consider the model that $\zeta(\boldsymbol{\lambda}_i(\boldsymbol{\beta}; \mathbf{Y}_i, A_i, F_i, q_i, r_i))$ is the conditional distribution of $\mathbf{Y}_i$ given $\mathbf{X}_i$ for $0 \leq i < n$. Then the weighted log-likelihood function is

$$\ell(\boldsymbol{\beta}) = \sum_{i=0}^{n-1} w_i \ell_c(\boldsymbol{\lambda}_i(\boldsymbol{\beta}); \mathbf{Y}_i, A_i, F_i, q_i, r_i). \tag{1}$$

If all w_i are 1, then $\ell(\boldsymbol{\beta})$ is the log-likelihood function. The weights w_i arise in complex sampling and in cases with repeat observations.

It follows that the gradient of ℓ at $\boldsymbol{\beta}$ in O is

$$\nabla\ell(\boldsymbol{\beta}) = \sum_{i=0}^{n-1} w_i \mathbf{X}_i^T \nabla\ell_c(\boldsymbol{\lambda}_i(\boldsymbol{\beta}); \mathbf{Y}_i, A_i, F_i, q_i, r_i), \tag{2}$$

and the Hessian matrix of ℓ at $\boldsymbol{\beta}$ is

$$\nabla^2\ell(\boldsymbol{\beta}) = \sum_{i=0}^{n-1} w_i \mathbf{X}_i^T \nabla^2\ell_c(\boldsymbol{\lambda}_i(\boldsymbol{\beta}); \mathbf{Y}_i, A_i, F_i, q_i, r_i)\mathbf{X}_i. \tag{3}$$

The Hessian matrix $\nabla^2\ell(\boldsymbol{\beta})$ has the approximation

$$\tilde{\nabla}^2\ell(\boldsymbol{\beta}) = -\sum_{i=0}^{n-1} w_i \mathbf{X}_i^T \nabla\ell_c(\boldsymbol{\lambda}_i(\boldsymbol{\beta}); \mathbf{Y}_i, A_i, F_i, q_i, r_i)[\nabla\ell_i(\boldsymbol{\lambda}_i(\boldsymbol{\beta}); \mathbf{Y}_i, A_i, F_i, q_i)]^T \mathbf{X}_i \tag{4}$$

(Haberman, 2013; Louis, 1982) .

The functions $\ell_c(\cdot;\mathbf{y},A,F,q,r)$ used are considered in this section. Some are examined in the literature on survival analysis (Cox, 1972; Kalbfleisch & Prentice, 2002), generalized linear models (McCullagh & Nelder, 1989), multivariate analysis (Anderson, 2003), and discrete choice (McFadden, 1973). It should be noted that names for models are somewhat variable in different references, especially for graded and cumulative cases. In addition, graded and cumulative cases are defined to be consistent with the Bernoulli cases. The following C++ functions are employed for common examples. The structs *f2v* are defined as in maxlinq2. If the argument *beta* is not in O, then all values returned are NaN. It is assumed that the user of the function has verified that the input vector y is in $\mathcal{Y}_i$. In the cases under study in this section, unless otherwise stated, the components are strictly concave, so that ℓ is strictly concave whenever $\mathbf{X}_i$, $0 \le i < n$, spans a space of dimension p. Conditions for a unique $\hat{\boldsymbol{\beta}}$ in O such that $\ell(\hat{\boldsymbol{\beta}})$ equals the supremum of ℓ over O are relatively complex (Haberman, 1974, 1977, 1980). It is worth noting that in cases in which $\hat{\boldsymbol{\beta}}$ in O satisfies the conditions that $\nabla\ell(\hat{\boldsymbol{\beta}})$ is the p-dimensional vector $\mathbf{0}_p$ with all elements 0 and $\nabla^2\ell(\hat{\boldsymbol{\beta}})$ is negative definite, then O can be restricted to ensure that ℓ is strictly concave on O and $\hat{\boldsymbol{\beta}}$ is the only member of O such that $\ell(\hat{\boldsymbol{\beta}})$ equals the supremum of ℓ on O and, for $\boldsymbol{\beta}$ in O, $\nabla\ell(\boldsymbol{\beta})$ is the unique vector with all elements 0 if $\boldsymbol{\beta}$ equals $\hat{\boldsymbol{\beta}}$. In all component functions, *order* is less than 1 if only the component value is computed, 1 if the component value and gradient are found, and greater than 1 if the component value, gradient, and Hessian are found. If *order* exceeds 2, the approximation of the Hessian by Equation 4 is employed. Repeated use is made of the struct *resp* with *ivec* component *iresp* and *vec* component *dresp*. In typical cases, *resp.dresp* or *resp.iresp* has no elements; however, exceptions do exist.

berresp

The function *berresp* is used to handle standard models for Bernoulli random variables. Here $q = r = 1$, A is S, $\mathcal{Y}(A,F,q,r)$ is the set of one-dimensional vectors $\mathbf{y}$ with element y_0 equal 0 or 1, and $O(A,F,q,r)$ is the set of all one-dimensional vectors, and F is in $\mathcal{F}$. For $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$ and $\boldsymbol{\beta}$ in $O(A,F,q,r)$ with element β_0,

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r) = \begin{cases} \log(F(\beta_0)), & y_0 = 1, \\ \log(1-F(\beta_0)), & y_0 = 0. \end{cases} \tag{5}$$

The function declaration is

*f2v berresp(const int & order, const char & transform, const resp & y,
const vec & beta).*

If *transform* is G, then $F = G_L$. If *transform* is H, then $F = G_U$. If *transform* is L, then $F = \Psi$. If *transform* is N, then $F = \Phi$. The function *berresp.value* is

$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

The function *berresp* requires loglogl, loglogu, logit, and probit.

betad

The function *betad* computes the function value, gradient, and Hessian matrix associated with the beta distribution with prameters $\exp(\beta_0)$ and $\exp(\beta_1)$. Here $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ has A with value B, $q=2$, $r=1$, $\mathcal{Y}(A,F,q,r)$ the set of positive real numbers less than 1, and $O(A,F,q,r)$ the set of two-dimensional vectors $\boldsymbol{\beta}$. The function declaration is

f2v betad(const int & order, const resp & y, const vec & beta).

Let the Beta function $B(a,b)$ with positive parameters a and b be $\Gamma(a)\Gamma(b)/\Gamma(a+b)$. If *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$, then the function *betad.value* is

$$\begin{aligned}\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r) &= \log(B(\exp(\beta_0),\exp(\beta_1))\\ &\quad+[\exp(\beta_0)-1]\log(y_0)+[\exp(\beta_1)-1]\log(1-y_0).\end{aligned}$$

The parametrization used has the advantage that the parameters β_0 and β_1 are unrestricted; however, $\ell_c(\cdot;\mathbf{y},A,F,q,r)$ is not concave.

contresp

The function *contresp* computes the function value, gradient, and Hessian matrix associated with the distribution of a location and scale model for a continuous random vector. Here $r=1$, $q=2$, A is D, $\mathcal{Y}(A,F,q,r)$ is the set of all one-dimensional vectors, $O(A,F,q,r)$ is the set R^2 of all two-dimensional vectors $\boldsymbol{\beta}$, and F is in $\mathcal{F}$. For $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$ and $\boldsymbol{\beta}$ in $O(A,F,q,r)$,

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)=\beta_1+\log(F_1(\beta_0+\exp(\beta_1)y_0)). \tag{6}$$

These cases correspond to a model that a random variable Y has a distribution function $F(\beta_0+\exp(\beta_1)y)$, where F is the distribution function of a random variable Z.

For all cases, the function declaration is

*f2v contresp(const int & order, const char & transform, const resp & y,
const vec & beta).*

The variable *transform* is defined as in berresp. The function *contresp.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. The function *contresp* requires gumbell, gumbelu, logistic, and normal.

The parametrtization does not lead to strict concavity unless β_1 is fixed.

cumresp

The function *cumresp* computes the function value, gradient, and Hessian matrix associated with a cumulative response model Here $r = 1$, $q \geq 1$, A is C, F is in $\mathcal{F}$, $\mathcal{Y}(A, F, q, r)$ is the set of one-dimensional vectors $\mathbf{y}$ such that y_0 is a nonnegative integer no greater than q, $O(A, F, q, r)$ is the set of all vectors of dimension q, and F is defined as in berresp. For $\boldsymbol{\beta}$ in $O(A, F, q, r)$ and $\mathbf{y}$ in $\mathcal{Y}(A, F, q, r)$, if $y_0 = k$, then

$$\ell_i(\boldsymbol{\beta}; \mathbf{y}) = \begin{cases} \log(1 - F(\beta_0), & k = 0, \\ \log(1 - F(\beta_k)) + \sum_{j=0}^{k-1} \log(F(\beta_j)), & 0 < k < q, \\ \sum_{j=0}^{q-1} \log(F(\beta_j)), & k = q. \end{cases} \tag{7}$$

The function declaration is

f2v cumresp(const int & order, const char & transform, const resp & y, const vec & beta).

Here *transform* is defined as in berresp. The function *cumresp.value* is $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. The function cumresp requires berresp, loglogl, loglogu, logit, and probit. If $r = 1$, then use of cumresp is equivalent to use of berresp. In general, $\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ is concave. Strict concavity holds if $q - y_0$ does not exceed 1.

dirichlet

The function *dirichlet* computes the function value, gradient, and Hessian matrix associated with a Dirichlet distribution with a q-dimensional parameter vector with elements $\exp \beta_j$, $0 \leq j \leq q - 1$, for the vector $\boldsymbol{\beta}$ of dimension q. Here $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ has A with value E, $\mathcal{Y}(A, F, q, r)$ the set of q-dimensional vectors with positive elements that sum to 1, and $O(A, F, q, r)$ is the set R^q of q-dimensional vectors. If $r = 1$, then the first element associated with the Dirichlet distribution has a beta distribution with parameters $\exp(\beta_0)$ and $\exp(\beta_1)$. The function declaration is

f2v dirichlet(const int & order, const resp & y, const vec & beta).

Let $z(\boldsymbol{\beta})$ be the sum of $\exp(\beta_j)$ over j from 0 to r. If *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$, the function *dirichlet.value* is

$$\begin{aligned} \ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r) &= \log(\Gamma(z(\boldsymbol{\beta}))) \\ &\quad + \sum_{j=0}^{r} \{\exp(\beta_j) - 1] \log(y_j) - \log(\Gamma(\exp(\beta_j)))\}. \end{aligned}$$

As in the case of betad, the parametrization leaves $\boldsymbol{\beta}$ unrestricted at the cost of loss of concavity.

gammad

The function *gammad* provides the computations required for $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ for the gamma distribution with A with value H, $q=2$, $r=1$, $\mathcal{Y}(A,F,q,r)$ the set of positive numbers, and $O(A,F,q,r)$ the set R^2 of two-dimensional vectors $\boldsymbol{\beta}$. The function declaration is

f2v gamma(const int & order, const resp & y, const vec & beta).

The function *gammad.value* is then

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)=\exp(\beta_1)[\log(y_0)+\beta_0]-\log(y_0)-\exp(\beta_0)y_0-\log(\Gamma(\exp(\beta_1)))$$

if *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. The exponent of β_0 is the scale parameter and the exponent of β_1 is the Gamma parameter. The vector $\boldsymbol{\beta}$ is unrestricted, but concavity does not hold, except that if β_1 is restricted to be 0 then strict concavity applies.

gradresp

The function *gradresp* computes the function value, gradient, and Hessian matrix associated with a graded response model Then $r=1$, $q\geq 1$, A is G, F is in $\mathcal{F}$, $O(A,F,q,r)$ is the set of all vectors of dimension q with strictly decreasing elements, $\mathcal{Y}(A,F,q,r)$ is the set of one-dimensional vectors $\mathbf{y}$ with $y_0=k$ a nonnegative integer no greater than q, and, for $\boldsymbol{\beta}$ in $O(A,F,q,r)$ and $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$,

$$\ell_c(\boldsymbol{\beta};\mathbf{y})=\begin{cases}\log(1-F(z_k(\beta_0))), & k=0,\\ \log(F(\beta_{k-1})-F(\beta_k)), & 0<k<q,\\ \log(F(\beta_{k-1})), & k=q.\end{cases}\tag{8}$$

The function declaration is

f2v gradresp(const int & order, const char & transform, const resp & y,
const vec & beta).

Here *transform* is defined as in berresp. The function *gradresp.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. If $q=1$, then berresp, cumresp and *gradresp* yield the same result. Required functions are nanf2v and berresp and its required functions.

gumbell

The function *gumbell* provides the computations required in contresp for $\ell_c(\cdot;\mathbf{y},A,F,q,r)$ for the minimum Gumbel case of $F=G_L$, A with value D, $q=2$, $r=1$, $\mathcal{Y}(A,F,q,r)$ the set of real numbers, and $O(A,F,q,r)$ the set of two-dimensional vectors $\boldsymbol{\beta}$ with $\beta_1>0$. The function declaration is *f2v gumbell(const int & order, const resp & y, const vec & beta).*

The function *gumbell.value* is then $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

gumbelu

The function *gumbelu* provides the computations required in contresp for $\ell_c(\cdot;\mathbf{y},A,F,q,r)$ for the maximum Gumbel case of $F = G_U$, A with value D, $q = 2$, $r = 1$, $\mathcal{Y}(A,F,q,r)$ the set of real numbers, and $O(A,F,q,r)$ the set of two-dimensional vectors $\boldsymbol{\beta}$ with $\beta_1 > 0$. The function declaration is

f2v gumbelu(const int & order, const resp & y, const vec & beta).

The function *gumbelu.value* is then $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

lcumresp

The function *lcumresp* computes the function value, gradient, and Hessian matrix associated with a cumulative response model with a slope parameter. Here $r = 1$, $q \geq 2$, A is c, F is in $\mathcal{F}$, $O(A,F,q,r)$ is the set of all vectors $\boldsymbol{\beta}$ of dimension q with β_j strictly decreasing for $0 \leq j < q-2$, $\mathcal{Y}(A,F,q,r)$ is the set of one-dimensional vectors $\mathbf{y}$ with $y_0 = k$ a nonnegative integer no greater than $q-1$, and, for $\boldsymbol{\beta}$ in $O(A,F,q,r)$ and $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$,

$$\ell_i(\boldsymbol{\beta};\mathbf{y}) = \begin{cases} \log(1 - F(\beta_0 + \beta_{q-1}), & k = 0, \\ \log(1 - F(\beta_k)) + \sum_{j=0}^{k-1} \log(F(\beta_j)), & 0 < k < q-1, \\ \sum_{j=0}^{q-1} \log(F(\beta_j + \beta_{q-1})), & k = q-1. \end{cases} \tag{9}$$

The function declaration is

f2v lcumresp(const int & order, const char & transform, const resp & y,
const vec & beta).

Here *transform* is defined as in berresp. The function *lcumresp.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. Required functions are ldscore, linsel, cumresp, and all functions required by cumresp.

lgradresp

The function *lgradresp* computes the function value, gradient, and Hessian matrix associated with a graded response transformation with a slope parameter. Here $r = 1$, $q \geq 2$, A is g, F is in $\mathcal{F}$, $O(A,F,q,r)$ is the set of all vectors $\boldsymbol{\beta}$ of dimension q with β_j strictly decreasing for $0 \leq j < q-1$, $\mathcal{Y}(A,F,q,r)$ is the set of one-dimensional vectors $\mathbf{y}$ with $y_0 = k$ a nonnegative integer no greater than $q-1$, and, for $\boldsymbol{\beta}$ in $O(A,F,q,r)$ and $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$,

$$\ell_c(\boldsymbol{\beta};\mathbf{y}) = \begin{cases} \log(1-F(z_k(\beta_0+\beta_{q-1}))), & k=0, \\ \log(F(\beta_{k-1}+\beta_{q-1})-F(\beta_k+\beta_{q-1})), & 0<k<q-1, \\ \log(F(\beta_{q-2}+\beta_{q-1})), & k=q-1. \end{cases} \tag{10}$$

The function declaration is

f2v lgradresp(const int & order, const char & transform, const resp & y, const vec & beta).

Here *transform* is defined as in berresp. The function *lgradresp.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. Required functions are nanf2v, ldscore, linsel, gradresp, and all functions required by gradresp. If $q=2$, *lgradresp* and lcumresp are equivalent.

lmultlogit

The function *lmultlogit* computes the function value, gradient, and Hessian matrix associated with a multinomial logit model with a slope parameter. In this case, $r=1$, $q\geq 2$, F is irrelevant, A is *l*, $\mathcal{Y}(A,F,q,r)$ is the set of one-dimensional vectors $\mathbf{y}$ such that y_0 is a nonnegative integer no greater than $q-1$, and $O(A,F,q,r)$ is the set of all q-dimensional vectors. For $\boldsymbol{\beta}$ in $O(A,F,q,r)$ and $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$, if $y_0=k$, then

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r) = \begin{cases} -\log\left(1+\sum_{k=0}^{q-2}\exp(\beta_k+(k+1)\beta_{q-1}\right), & k=0, \\ \beta_{k-1}+k\beta_{q-1}+\ell_c(\boldsymbol{\beta};\mathbf{0}_1,A,F,q,r), & 0<k<q-1. \end{cases} \tag{11}$$

The function declaration is

f2v lmultlogit(const int & order, const resp & y, const vec & beta).

The function *lmultlogit.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. If $q=1$, use of *lmultlogit* gives the same result as use of lcumresp and as use of lgradresp with*transform* equal *L*. The function *lmultlogit* requires lscore, namereflinsel, and multlogit.

logistic

The function *logistic* provides the computations required in contresp for $\ell_c(\cdot;\mathbf{y},A,F,q,r)$ for the logistic case $F=\Psi$ in contresp with A with value *D*, $q=2$, $r=1$, $\mathcal{Y}(A,F,q,r)$ the set of real numbers, and $O(A,F,q,r)$ the set of two-dimensional vectors $\boldsymbol{\beta}$ with $\beta_1>0$. The function declaration is *f2v logistic(const int & order, const resp & y, const vec & beta).*

The function *logistic.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

logit

The function *logit* computes the function value, gradient, and Hessian matrix associated with the logit case in berresp with A equal to S, $F=\Psi$, and $q=r=1$. The function declaration is

f2v logit(const int & order, const resp & y, const vec & beta).

The function *logit.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

loglogl

The function *loglogl* computes the function value, gradient, and Hessian matrix associated with the complementary log-log case of berresp with A equal to S, $F=G_L$, and $q=r=1$. The function declaration is

f2v loglogl(const int & order, const resp & y, const vec & beta).

The function *loglogl.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

loglogu

The function *loglogu* computes the function value, gradient, and Hessian matrix associated with the log-log case of berresp with A equal to S, $F=G_U$, and $q=r=1$. The function declaration is

f2v loglogu(const int & order, const resp & y, const vec & beta).

The function *loglogu.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

logmean

The function *logmean* computes the function value, gradient, and Hessian matrix associated with a log-mean transformation for a Poisson random variable. In this case, $q=r=1$, A is P, the value of F in $\mathcal{F}$ is irrelevant, $\mathcal{Y}(A,F,q,r)$ is the set of one-dimensional vectors $\mathbf{y}$ such that y_0 is a nonnegative integer, and $O(A,F,Q,R)$ is the set of all one-dimensional vectors. For $\boldsymbol{\beta}$ in $O(A,F,q,r)$ and $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$,

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)=y_0\beta_0-\exp(\beta_0)-\log([y_0]!). \tag{12}$$

The function declaration is

f2v logmean(const int & order, const resp & y, const vec & beta).

The function *logmean.value* is $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

maxberresp

The function *maxberresp* finds the log likelihood component, gradient, and Hessian matrix for the maximum of two unobserved Bernoulli random variables. Here $q = 2$, $r = 1$, A is M, F is in $\mathcal{F}$, $O(A, F, q, r)$ is the set of two-dimensional vectors, and $\mathcal{Y}(A, F, q, r)$ is the set of one-dimensional vectors $\mathbf{y}$ with y_0 equal 0 or 1. For y in $\mathcal{Y}(A, F, q, r)$ and $\boldsymbol{\beta}$ in $O(A, F, q, r)$,

$$\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r) = \begin{cases} \log(F(\beta_0) + F(\beta_1) - F(\beta_0)F(\beta_1)), & y_0 = 1, \\ \log(1 - F(\beta_0)) + \log(1 - F(\beta_1)), & y_0 = 0. \end{cases} \tag{13}$$

It should be noted that

$$F(\beta_0) + F(\beta_1) - F(\beta_0)F(\beta_1) = 1 - [1 - F(\beta_0)][1 - F(\beta_1)] \tag{14}$$

and

$$\log(1 - F(\beta_0)) + \log(1 - F(\beta_1)) = \log([1 - F(\beta_0)][1 - F(\beta_1)]). \tag{15}$$

The function $\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ is not necessarily concave if $y_0 = 1$.

The function declaration is

f2v maxberresp(const int & order, const char & transform, const resp & y, const vec & beta).

The variable *transform* is defined as in berresp. The function *maxberesp.value* is $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.The functions berresp, logit, loglogl, loglogu, and probit are required.

multlogit

The function *multlogit* computes the function value, gradient, and Hessian matrix associated with a multinomial logit transformation. In this case, $r = 1$, $q \geq 1$, F is irrelevant, A is L, $\mathcal{Y}(A, F, q, r)$ is the set of one-dimensional vectors $\mathbf{y}$ such that y_0 is a nonnegative integer no greater than q, and $O(A, F, q, r)$ is the set of all q-dimensional vectors. For $\boldsymbol{\beta}$ in $O(A, F, q, r)$ and $\mathbf{y}$ in $\mathcal{Y}(A, F, q, r)$, if $y_0 = k$, then

$$\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r) = \begin{cases} -\log\left(1 + \sum_{k=0}^{q-1} \exp(\beta_k\right), & k = 0, \\ \beta_{k-1} + \ell_c(\boldsymbol{\beta}; \mathbf{0}_1, A, F, q, r), & k > 0. \end{cases} \tag{16}$$

The function declaration is

f2v multlogit(const int & order, const resp & y, const vec & beta).

The function *multlogit.value* is $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. If $q = 1$, use of multlogit gives the same result as use of logit and as use of berresp, cumresp, or gradresp with *transform* equal L.

normal

The function *normal* computes the function value, gradient, and Hessian matrix associated with the normal case in contresp. Thus A is D, $F = \Phi$, $q = 2$, $r = 1$, $\mathcal{Y}(A, F, q, r)$ is the space of one-dimensional vectors, and $O(A, F, q, r)$ is the set of two-dimensional vectors $\boldsymbol{\beta}$. The function declaration is

f2v normal(const int & order, const vec & y, const vec & beta).

The function *normal.value* is $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ if *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

probit

The function *probit* computes the function value, gradient, and Hessian matrix associated with a probit transformation in berresp with A equal to S, $F = \Psi$, and $q = r = 1$. The function declaration is

f2v probit(const int & order, const resp & y, const vec & beta).

The function *probit.value* is $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$.

ranklogit

The function *ranklogit* computes the function value, gradient, and Hessian matrix associated with a model for discrete choice in which $q + 1$ objects are ranked for some positive integer q and the r most-preferred objects are recorded for some positive integer $r \leq q$. Here A has value R, F is irrelevant, the set $\mathcal{Y}(A, F, q, r)$ consists of the vectors $\mathbf{y}$ of dimension r with distinct nonnegative integer elements that are no greater than q, and $O(A, F, q, r)$ is the set of all q-dimensional vectors. To describe the model, consider $\boldsymbol{\beta}$ in $O(A, F, q, r)$. Let U_j, $0 \leq j \leq q$, be independent random variables such that U_0 and $U_j - \beta_j$, $1 \leq j \leq q$, have the common distribution function G_U. Let υ denote a random permutation of the integers from 1 to q such that $U_{\upsilon(j)}$, $1 \leq j \leq q$, is nonincreasing in j. For $\boldsymbol{\beta}$ in $O(A, F, q, r)$ and $\mathbf{y}$ in $\mathcal{Y}(A, F, q, r)$, the log-likelihood $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ is the logarithm of the probability that $\upsilon^{-1}(k) = y_k$ for $0 \leq k \leq r$. To find $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$, let $\boldsymbol{\alpha}(\boldsymbol{\beta})$ be the vector of dimension $q + 1$ such that element j, $0 \leq j \leq q$, is $\alpha_j(\boldsymbol{\beta}) = 0$ if $j = 0$ and $\alpha_j(\boldsymbol{\beta}) = \beta_{j-1}$ if $j > 0$. Let $K_j(\mathbf{y})$ be the set of nonnegative integers no greater than q that are not equal to y_k for any nonnegative integer $k < j$. Thus $K_0(\mathbf{y})$ is the set of nonnegative integers no greater than q. Then the log-likelihood component is

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)=\sum_{j=0}^{r-1}\left\{\alpha_{y_j}(\boldsymbol{\beta})-\log\left[\sum_{k\in K_j(\mathbf{y})}\exp(\alpha_k(\boldsymbol{\beta}))\right]\right\}. \tag{17}$$

The function declaration is

f2v ranklogit(const int & order, const resp & y, const vec & beta).

The function *ranklogit.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. If $r=1$, use of *ranklogit* gives the same result as use of multlogit.

truncresp

The function *truncresp* computes the function value, gradient, and Hessian matrix associated with a right-censored continuous random variable with the distribution of $\beta_0+\beta_1 Z$ for some real β_0 and positive real β_1, where, as in contresp, Z has distribution function F in $\mathcal{F}$. In this case, $q=r=2$, A is T, $\mathcal{Y}(A,F,q,r)$ consists of two-dimensional vectors $\mathbf{y}$ such that y_0 is a real number and y_1 is 0 or 1, and $O(A,F,q,r)$ is the set of all two-dimensional vectors $\boldsymbol{\beta}$ with element $\beta_1>0$. For $\boldsymbol{\beta}$ in $O(A,F,q,r)$ and $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$, if $y_1=0$, then the observation is not censored and the corresponding log-likelihood component is

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)=\log(\beta_1)+\log(F_1(\beta_0+\beta_1 y_0)), \tag{18}$$

while in the case of $y_1=1$, the the observation is censored at y_0 and the log-likelihood component is

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)=\log(1-F(\beta_0+\beta_1 y_0)). \tag{19}$$

The function declaration is

f2v truncresp(const int & order, const char & transform, const resp & y,
const vec & beta).

Here *y.iresp* has the single element y_1, *y.dresp* has the single element y_0, *beta* is $\boldsymbol{\beta}$, *transform* is defined as in berresp, and *truncresp.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$. Functions required are berresp, contresp, and their respective required functions.

Computation of Log Likelihood Functions

genresp

The function *genresp* provides a general tool for computation of a a component of a log-likelihood function, its gradient, and its Hessian matrix. The function declaration is

f2v genresp(const int & order, const model & choice, const resp & y,
const vec & beta).

Here *model* has the definition

struct model{char type; char transform}.

In *choice*, *choice.type* has value B for the beta case of betad, C for cumulative case of cumresp, D for the continuous case of contresp, E for the Dirichlet case of dirichlet, G for the graded response case of gradresp, H for the gamma case of gammad, L for the multinomial logit case of multlogit, M for the maximum of two independent Bernoulli variables described in maxberresp, P for the log-mean Poisson case of logmean, R for the rank-logit case of ranklogit, S for the Bernouli case of berresp, and T for the censored continuous case of truncresp. For discrete cases, *choice.transform* has possible values G for the complementary log-log case of loglogu, H for the log-log case of loglogl, L for the logit case of logit, and N for the probit case of probit. For continuous cases, G is for the minimum Gumbel distribution of gumbell, H is for the maximum Gumbel case of gumbelu, L is for the logistic case of logistic, and N is for the normal case of normal. For example, *choice.type* is C and *choice.transform* is G for the cumulative complementary log-log case, while *choice.type* is S and *choice.type* is N in the probit case. The variable *choice.transform* is only relevant if *choice.type* is C, D, G, M, S, or T.

The function *genresp* uses berresp, contresp, cumresp, gradresp, gammad, betad, dirichlet, logmean, maxberresp, multlogit, ranklogit, and truncresp, together with the functions they in turn require.

genresplik

The function *genresplik* computes the log-likelihood function and its gradient and Hessian matrix. The function declaration is

f2v genresplik(const int & order, const field<pattern> & patterns,
const xsel & patternnumber, const field<resp> & data,
const field<xsel> & selectbeta, const xsel & selectbetano,
const vec & w, const xsel & obssel, const vec & beta).

The definitions of arguments and structs rely the the definition of *model* in genresp and the definition of *resp* in truncresp. In the case of *xsel*, the struct is defined as

struct xsel{bool all; uvec list}.

The struct is applied to a finite and nonempty collection collection of k objects numbered from 0 to $k-1$. If *all* is *true*, all members of the collection are considered.

Otherwise, only members in *list* are used. Thus *obssel* specifies the observation numbers i to be used. The function intsel is used for selections based on *xsel*. If *xsel.all* is *true*, then an integer i is mapped by intsel to i, whereas if *xsel.all* is *false*, then i is mapped to the ith element of *xsel.list*.

If *patternnumber* maps the integer i to $j(i,1)$, then element $j(i,1)$ of *patterns* defines the function $\boldsymbol{\lambda}_i$ and the model used for the response $\mathbf{Y}_i$. In addition, the weight w_i assigned to $\mathbf{Y}_i$ is element i of *w*. If *selectbetano* maps i to $j(i,2)$, element $j(i,2)$ of *selectbeta* specifies the elements of *beta* used to define $\boldsymbol{\lambda}_i$.

The struct *pattern* is defined as

struct pattern{model choice; vec o; mat x;}.

For element $j(i,1)$ of *patterns*, the component *choice* defines the model for $\mathbf{Y}_i$, *o* defines $\mathbf{o}_i$, while *x* is used to specify $\mathbf{X}_i$. If k is not selected by element $j(i,2)$ of *selectbeta*, then column k of $\mathbf{X}_i$ is the zero vector. Otherwise, let k be the mth element selected by element $j(i,2)$ of *selectbeta*. Then column m of component x becomes column k of $\mathbf{X}_i$. Element i of *data* defines $\mathbf{Y}_i$. The dimension of $\mathbf{Y}_i$ is the sum of the dimensions of the *iresp* and *dresp* components of the ith element of *data*. *genresplik* uses addsel, intsel, sintsel, sivecsel, svecsel, vecsel, genresp, and all C++ functions genresp requires.

genrespmle

The function *genrespmle* maximizes the log-likelihood function. It uses the function, gradient, and Hessian matrix of genresplik. The function declaration is

maxf2v genrespmle(const int & order, const params & mparams,
const char & algorithm, const field<pattern> & patterns,
const xsel & patternnumber, const field<resp> & data,
const field<xsel> & selectbeta, const xsel & selectbetano,
const vec & w, const xsel & obssel, const vec & start).

Definitions are as in maxlinq2, maxf2vvar, maxselect, and genresplik. The functions maxselect and genresplik are required, together with all C++ functions that these two functions need.

Tools for Computation of Log Likelihood Functions

addsel

The function *addsel* is used to add *f2v* structures. The function declaration is

void addsel(const int & order, const xsel & xselect,
const f2v & x, f2v & y, const double & a).

Here *order* and *xselect* are defined as in Log-likelihood Components. The struct *y* is modified by use of the struct *x* and the multiplier *a*. In all cases, *ax.value* is added to *y.value*. If *xselect.all* is *true*, then *x* and *y* have compatible dimensions, *ax.grad* is added to *y.grad* if *order* is at least 1, and *ax.hess* is added to *y.hess* if *order* is at least 2. If *xselect.all* is *false*, then *ax.grad* is added to *y.grad.elem(xselect.list)* if *order* is at least 1 and *ax.hess* is added to *y.hess.submat(xselect.list,xselect.list)* if *order* is at least 2.

dupname

The function *dupname* checks if a vector of names contains any duplicates. The function declaration is

bool dupname(const vector<string> & names).

The function value is *true* if *names* has at least two equal elements and *false* otherwise.

intsel

The function *ivecsel* selects a nonnegative integer from a struct *xsel* defined as in genresplik. The function declaration is

int intsel(const xsel & xselect, const int & i).

The value of *intsel* is the ith integer defined by *xselect*.

ivecsel

The function *ivecsel* is employed to create a new integer vector from an old vector of integers by extracting of elements of the old vector. The function declaration is

ivec ivecsel(const xsel & xselect, const ivec & y).

Here the struct *xsel* is defined as in genresplik. If *xselect.all* is *true*, then *ivecsel* is *y*. Otherwise, *ivecsel* is a vector with the number of elements in component *list* of *xselect*, if $j(i)$ is element i of of component *list* of *xselect*, then element i of *ivecsel* is element $j(i)$ of *y*.

ldscore

For an integer $k > 1$, the function *ldscore* produces a $k-1$ by k matrix with the first $k-1$ columns equal to the $k-1$ by $k-1$ identity matrix and the last column with all elements 1. The function declaration is *mat ldscore(const int & k).*

linsel

The function *linsel* is used to apply a linear transformation to an *f2v* struct. The function declaration is

f2v linsel(const int & order, const f2v & x, const mat & a).

Here *order* is defined as in Log-likelihood Components. Let **a** be the matrix defined by *a*. It is always the case that *linsel.value* is *x.value*. If *order* is positive and **g** is the vector specified by *x.grad*, then *linsel.grad* is $\mathbf{a}'\mathbf{g}$. If *order* exceeds 1 and **H** is the matrix specified by *x.hess*, then *linsel.hess* is $\mathbf{a}'\mathbf{H}\mathbf{a}$.

lscore

For an integer $k > 1$, the function *lscore* produces a $k-1$ by k matrix with the first $k-1$ columns equal to the $k-1$ by $k-1$ identity matrix and the last column with element j equal to $j+1$ for $0 \leq\leq k-1$. The function declaration is

mat lscore(const int & k).

nanf2v

The function *nanf2v* returns a *f2v* struct with all elements with value *NaN*. The function declaration is

f2v nanf2v(const int & order, const f2v & x).

The struct *f2v* is defined as in maxlinq2. The value of *nanf2v.value* is always set to *NaN*. If *order* is positive, *nanf2v.grad* and *x.grad* have the same number of elements, and all elements of *nanf2v.grad* are *NaN*. If *order* exceeds 1, then *nanf2v.hess* and *x.hess* have the same size and all elements of *nanf2v.hess* are *NaN*.

sintsel

The function *sintsel* counts the number of integers selected from the first n nonnegative integers. The function declaration is

int sintsel(const xsel & xselect, const int & n).

The integers from 0 to $n-1$ are selected according to *xselect*, where *xsel* is defined as in genresplik.

sivecsel

The function *sivecsel* counts the number of integer vector elements selected. The function declaration is

int sivecsel(const xsel & xselect, const ivec & y).

The integers in *y* are selected according to *xselect*, where *xsel* is defined as in genresplik.

svecsel

The function *svecsel* counts the number of vector elements selected. The function declaration is

int svecsel(const xsel & xselect, const vec & y).

The integers in *y* are selected according to *xselect*, where *xsel* is defined as in genresplik.

vecsel

The function vecsel is employed to create a new vector from an old vector by extracting of elements of the old vector. The function declaration is

vec vecsel(const xsel & xselect, const vec & y).

Here the struct *xsel* is defined as in genresplik. If *xselect.all* is *true*, then *vecsel* is *y*. Otherwise, *vecsel* is a vector with the number of elements in *xselect.list*, and element i of *vecsel* is element $j(i)$ of *y* if $j(i)$ is element i of component *list* of *xselect*.

Latent Structures

In this section, functions useful for analysis of latent structures are considered. The log-likelihood function in this section is defined based on the definitions in Log-likelihood Components; however, some of the variables are latent rather than observed. In typical cases, data involve multiple responses for each individual observation. For a positive integer m, m observations are present. For observation h, $0 \leq h < m$, the observation has weight $w_{h*} > 0$, and n_h responses are observed. For a positive integer n and an observation i, $0 \leq i < n$, positive integers q_i and r_i and character variables A_i in $\mathcal{A}$ and F_i in $\mathcal{F}$ are given. The component of the log likelihood for observation i involves the predicted random vector $\mathbf{Y}_i$ in $\mathcal{Y}(A_i, F_i, q_i, r_i)$, the q_i by p predicting matrix $\mathbf{X}_i$ in a nonempty set $\mathcal{X}_i$, the q_i-dimensional vector $\mathbf{o}_i$, and the positive real weight w_i. If $\boldsymbol{\beta}$ is in O, then let $\boldsymbol{\lambda}_i(\boldsymbol{\beta}) = \mathbf{o}_i + \mathbf{X}_i\boldsymbol{\beta}$ be in $O(A_i, F_i, q_i, r_i)$ for $0 \le i < n$, and let the log-likelihood function under study have the form

$$\ell(\boldsymbol{\beta}) = \sum_{i=0}^{n-1} w_i \ell_c(\boldsymbol{\lambda}_i(\boldsymbol{\beta}); \mathbf{Y}_i, A_i, F_i, q_i, r_i). \tag{20}$$

In addition, a latent vector appears in the model such the observed responses are conditionally independent given the latent vector and the predicting variables. Associated with the latent vector are positive integers q_* and r_*, A_* in $\mathcal{A}$, and F_* in $\mathcal{F}$. The latent vector $\boldsymbol{\theta}_h$ is in $\mathcal{Y}(A_*, F_*, q_*, r_*)$ and is predicted by the q_* by p predicting matrix $\mathbf{X}_{h*}$ in the nonempty set $\mathcal{X}_*$ and the fixed q_*-dimensional vector $\mathbf{o}_*$. It is assumed that $\boldsymbol{\lambda}_*(\boldsymbol{\beta}) = \mathbf{o}_* + \mathbf{X}_*\boldsymbol{\beta}$ is in $O(A_*, F_*, q_*, r_*)$ as long as $\mathbf{X}_*$ is in $\mathcal{X}_*$ and $\boldsymbol{\beta}$ is in O. For response i, $0 \le i < n_h$, positive integers q_{hi} and r_{hi} are given. The variable A_{hi} is in $\mathcal{A}$ and F_{hi} is in $\mathcal{F}$. The component of the log likelihood for response i involves the predicted random vector $\mathbf{Y}_{hi}$ in $\mathcal{Y}(A_{hi}, F_{hi}, q_{hi}, r_{hi})$, the latent vector $\boldsymbol{\theta}_h$, the q_{hi} by p predicting matrix $\mathbf{X}_{hi}$ in a nonempty set $\mathcal{X}_{hi}$, the q_{hi}-dimensional vector $\mathbf{o}_{hi}$, the q_{hi} by q_* matrix $\mathbf{D}_{hi}$, the positive real weight w_{hi}, the q_{hi} by p matrix $\mathbf{D}_{hik}$, $0 \le k < q_*$, and the function $\ell_c(\cdot; \mathbf{y}, A_{hi}, F_{hi}, q_{hi}, r_{hi})$ on $O(A_{hi}, F_{hi}, q_{hi}, r_{hi})$ defined for $\mathbf{y}$ in $\mathcal{Y}(A_{hi}, F_{hi}, q_{hi}, r_{hi})$. For any $\boldsymbol{\beta}$ in O, $\mathbf{X}$ in $\mathcal{X}_{hi}$, and $\boldsymbol{\theta}$ in $\mathcal{Y}(A_*, F_*, q_*, r_*)$,

$$\boldsymbol{\lambda}_{hi}(\boldsymbol{\beta}|\boldsymbol{\theta}) = \mathbf{o}_{hi} + \mathbf{X}_{hi}\boldsymbol{\beta} + \mathbf{D}_{hi}\boldsymbol{\theta} + \sum_{k=0}^{p-1} \theta_k \mathbf{D}_{hik}\boldsymbol{\beta} \tag{21}$$

is in $O(A_{hi}, F_{hi}, q_{hi}, r_{hi})$. A helpful general reference to applications in psychometrics is provided by van der Linden (2018). A large collection of references to other applications provided in Muthén (2002)

For example, in a simple two-parameter item-response model for dichotomous responses, a pool of T items numbered from 0 to $T-1$ is used, and $p = 2T$. The element β_{2t} is the item intercept for item t, and β_{2t+1} is the corresponding item slope. The response $\mathbf{Y}_{hi}$, which corresponds to pool member $t(h,i)$, is a one-dimensional vector with the single element Y_{0hi} equal to 0 or 1, the $t(h,i)$ are all different for the same observation h, each q_{hi} and each r_{hi} is 1, r_* is 1, q_* is 2, $\mathbf{o}_*$ is the two-dimensional vector with elements 0 and 1, all elements of the $\mathbf{X}_{h*}$ are 0, the only nonzero element of $\mathbf{X}_{hi}$ is row 0 and column $2t(h,i)$, which is 1, the $\mathbf{D}_{hk}$ have all 0 elements, the $\mathbf{D}_{hik}$ are 1 by 1 matrices with the single element 0 except for $\mathbf{D}_{hi(2t(h,i)+1)}$, which is the 1 by 1 matrix with the single element 1, and $\mathbf{o}_{hi}$ is the vector with the single element 0. Thus $\boldsymbol{\lambda}_{hi}(\boldsymbol{\beta}|\boldsymbol{\theta})$ is the vector with the single element $\beta_{2t(h,i)} + \beta_{2t(h,i)+1}\theta_0$. The value of A_{hi} is S, and each F_{hi} is the same. If each F_{hi} is Ψ, then the model is a two-parameter logistic (2PL) model. If each F_{hi} is Φ, then the model is a normal ogive (2PN) model.

For $\boldsymbol{\beta}$ in O, the log-likelihood has the form

$$\ell(\boldsymbol{\beta}) = \sum_{h=0}^{m-1} w_{h*} \ell_h(\boldsymbol{\beta}), \tag{22}$$

where $\ell_h(\boldsymbol{\beta})$ is the component of the log-likelihood for observation h. Thus the gradient function of ℓ at $\boldsymbol{\beta}$ satisfies

$$\nabla\ell(\boldsymbol{\beta}) = \sum_{h=0}^{m-1} w_{h*}\nabla\ell_h(\boldsymbol{\beta}), \tag{23}$$

where $\nabla\ell_h(\boldsymbol{\beta})$ is the gradient function of ℓ_h at $\boldsymbol{\beta}$. The Hessian function of ℓ at $\boldsymbol{\beta}$ satisfies

$$\nabla^2\ell(\boldsymbol{\beta}) = \sum_{h=0}^{m-1} w_{h*}\nabla^2\ell_h(\boldsymbol{\beta}), \tag{24}$$

where $\nabla^2\ell_h(\boldsymbol{\beta})$ is the Hessian function of ℓ_h at $\boldsymbol{\beta}$. The approximation

$$\tilde{\nabla}^2\ell(\boldsymbol{\beta}) = -\sum_{h=0}^{m-1} w_{h*}\nabla\ell_h(\boldsymbol{\beta})[\nabla\ell_h(\boldsymbol{\beta})]^T, \tag{25}$$

may also be considered.

In turn, $\ell_h(\boldsymbol{\beta})$ involves the product

$$\ell_h(\boldsymbol{\beta}|\boldsymbol{\theta}) = \ell_c(\boldsymbol{\lambda}_*(\boldsymbol{\beta});\boldsymbol{\theta}, A_*, F_*, q_*, r_*)\sum_{i=0}^{n_h-1} w_{hi}\ell_c(\boldsymbol{\lambda}_{hi}(\boldsymbol{\beta}|\boldsymbol{\theta});\mathbf{Y}_{hi}, A_{hi}, F_{hi}, q_{hi}, r_{hi}) \tag{26}$$

for $\boldsymbol{\theta}$ in $\mathcal{Y}(A_*, F_*, q_*, r_*)$. The component

$$\ell_h(\boldsymbol{\beta}) = \log\int(\exp(\ell_h(\boldsymbol{\beta}|\cdot))), \tag{27}$$

where $\exp(\ell_h(\boldsymbol{\beta}|\cdot))$ is the function with value $\exp(\ell_h(\boldsymbol{\beta}|\boldsymbol{\theta}))$ for $\boldsymbol{\theta}$ in $\mathcal{Y}(A_*, F_*, q_*, r_*)$. In practice, $\ell_h(\boldsymbol{\beta}))$ is evaluated by

$$\tilde{\ell}_h(\boldsymbol{\beta}) = \log\left[\sum_{k=1}^{Q} u_{hk}\exp(\ell_h(\boldsymbol{\beta}|\boldsymbol{\theta}_{hk}))\right], \tag{28}$$

for some positive weights u_{hk} and elements $\boldsymbol{\theta}_{hk}$ in $\mathcal{Y}(A_*, F_*, q_*, r_*)$.

For computations for latent-structure models, the following functions are employed.

gpcm

The main function *gpcm* finds maximum-likelihood estimates for a generalized partial credit (GPCM) model (Muraki, 1992) with a standard normal latent variable. As in the two-parameter item-response model for dichotomous responses, a pool of T items numbered from 0 to $T-1$ is used. For each nonnegative integer $t < T$, I_t is a positive integer greater than 1 that corresponds to pool member t, and p is the sum of the integers I_t for $0 \le t < T$. In the GPCM case, response $\mathbf{Y}_{hi}$ is a one-dimensional vector with the single element Y_{0hi}. The response corresponds to item $t(h,i)$ from the pool, where $t(h,i)$ is a nonnegative integers less than T. Item $t(h,i)$ has $I_{t(h,i)}$ possible nonnegative integer values from 0 to $I_{t(h,i)}-1$, the latent vector $\boldsymbol{\theta}_h$ has dimension 1, and θ_{h0} has a standard normal distribution. The $t(h,i)$ are all different for the same observation h. Let J_{hi0} be the sum of the I_u for $u < t(h,i)$, let J_{hi1} be the sum of the I_u for $u \leq t(h,i)$, and let the nonnegative integer k be less than $I_{t(h,i)}-1$. Conditional on θ_{h0}, Y_{0hi} has a multinomial logit distribution with element k of the parameter vector $\boldsymbol{\lambda}_{hi}(\boldsymbol{\beta}|\boldsymbol{\theta})$ equal to $\beta_a + \beta_b k\theta_{h0}$, where $a = J_{hi0}+k$ and $b = J_{hi1}$.

The function *gpcm* considers the simplified case of n_h constant and $t(h,i) = i$ for nonnegative integers $i < n_0$ and $I_i - 1$ equal to the maximum value of Y_{0hi} for $0 \leq h < m$. The function uses a control file *controlfile* read from standard input. The file is a text file with any line consisting of a pair of entries separated by a space, The entries contain no blank characters. The following cases exist.

The variable *data* has string value *infile* that specifies the input file for the data. The default is *infile.csv*. Data are read as an integer matrix with row h and column i equal to Y_{0hi}.

The variable *sf* is associated with the string *startvalue* that specifies a file containing the vector of starting values. The default is to apply startgpcm.

The variable *outfile* has string value that specifies the name of the output file for results. The default is *outfile*.

The bool variable *fflag* indicates whether an output file is used. The default is *true*.

The bool variable *pflag* indicates whether anything is printed in ascii form in standard output. The default is *true*.

The character variable *method* specifies the algorithm applied. The possibilities are G for gradient ascent, C for conjugate gradient ascent, N for modified Newton-Raphson, and L for Louis approximation. The default is *N*.

The positive double variable *tol* specifies the convergence criterion *mparams.tol*. The default is 0.001.

The bool variable *adapt* indicates whether adaptive quadrature is used. The default is *true*.

The character variable *quadrature* indicates the quadrature procedure used, with G for Gauss-Hermite and Q for normal quantiles. The default is *G*.

The positive integer variable *points* that exceeds 1 gives the number of quadrature points, with the default 9.

The program uses irtmle and all its required functions. In addition, startgpcm and its required functions, hermpw, qnormpwe, and savmaxf2v are needed.

invcdf

The function *invcdf* finds the inverse of the cumulative distribution function and its derivative at *prob* for F in $\mathcal{F}$. The function declaration is *f1 invcdf(const char & cdf, const double & prob).*

The struct *f1* is

struct f1{double value; double der}.

The variable *cdf* is defined as in berresp. At *prob*, the value of the inverse is *invcdf.value*, and the derivative is *invcdf.der*.

irtm

The function *irtm* finds the log likelihood component $\ell_h(\boldsymbol{\beta})$ and associated gradient and Hessian matrix for a latent structure model. The function uses numerical integration if $\mathcal{Y}(A_*, F_*, q_*, r_*)$ is not finite or countably infinite. The function declaration is

f2v irtm (const int & order, const field<pattern> & patterns,
const xsel & patternnumber, const field<resp> & data, const field <pwr> & thetas,
const adq & scale, dovecmat & obsscale,
const field<xsel> & selectbeta, const xsel & selectbetano,
const field<xsel> & selectbetac, const xsel & selectbetacno,
const field<xsel> & selectthetai, const xsel & selectthetaino,
const field<xsel> & selectthetad, const xsel & selectthetadno,
const field<xsel> & selectthetac, const xsel & selectthetacno,
const vec & w, const xsel & obssel, const vec & beta).

In this declaration, almost all arguments are defined as in genresplik. The exceptions are *thetas*, *scale*, and *obsscale*. These arguments rely on the structs *pwr*, *adq*, and *dovecmat* with the following definitions:

struct pwr{double weight; double kernel; resp theta;},

struct adq{bool adapt; xsel linselect;xselv quadselect;},

and

struct dovecmat{double s; vec v; mat m;}.

The structs *thetas* and *obsscale* define the u_{hk} and $\boldsymbol{\theta}_{hk}$ of Equation 28, while *scale* is used in irtmle for computation of maximum-likelihood estimates of $\boldsymbol{\beta}$. For element k of *thetas*, u_{hk} is obtained by multiplying *thetas(h).weight* by *obsscale.s* and dividing by *thetas(h).kernel*, the integer elements of $\boldsymbol{\theta}_{hk}$ are the same as in *thetas(h).theta.iresp*, and the remaining elements of $\boldsymbol{\theta}_{hk}$ are found by adding *obsscale.v* to the product of the matrix *obsscale.m* and the vector *thetas(h).theta.dresp*. Use of *scale* is discussed in the description of irtmle. In addition to functions needed by genresplik, fitquad and its required functions are needed.

irtmle

The function *irtmle* finds the maximum likelihood estimate for a latent structure model. As in irtm, the function uses numerical integration if $\mathcal{Y}(A_*, F_*, q_*, r_*)$ is not finite or countably infinite. The function declaration is

maxf2v irtmle (const int & order, const params & mparams,
const char & algorithm, const field<patterns> & patterns,
const field<xsel> & patternnumber, const xsel & patno,
const field<field<resp> > & data, const field<field <pwr> > & thetas,
const xsel & thetano,
const field<adq> & scale, const xsel & scaleno, field<dovecmat> & obsscale,
const field<xsel> & selectbeta, const field<xsel> & selectbetano,
const xsel & selbetano,
const field<xsel> & selectbetac, const field<xsel> & selectbetacno,
const xsel & selbetacno,
const field<xsel> & selectthetai, const field<xsel> & selectthetaino,
const xsel & selthetaino,
const field<xsel> & selectthetad, const field<xsel> & selectthetadno,
const xsel & selthetadno,
const field<xsel> & selectthetac, const field<xsel> & selectthetacno,
const xsel & selthetacno,
const field<vec> & w, const xsel & wno,
const field<xsel> & obssel, const xsel & obsselno,
const vec & obsweight, const xsel & datasel,
const field<xsel> & betasel, const xsel & betaselno, const vec & start).

In this declaration, *maxf2v* and *mparams* are defined as in maxlinq2 and maxf2vvar, while *order* is defined as in genresplik, whereas *mparams* is used for the basic iterations used for determination of the maximum-likelihood estimate, The variables *algorithm* and *start* are defined as in genrespmle. The vector *obsweight* provides the weights w_{h*} for $0 \le h \le m-1$. The variables *patterns*, *selectbeta*, *selectbetac*, *selectthetai*, *selecthetad*, and *selectthetac* are defined as in *genresplik*. For each observation h and each nonnegative integer $i < n_h$, if $m(h,1)$ is *patno(h)*, then *patternno(m(h,1))* assigns i to the member of *patterns* that corresponds to $\boldsymbol{\lambda}_{hi}$. Similar arguments apply to triples such as *selectbeta*, *selectbetano*, and *selbeta*. In the case of *w* and *wno*, for each observation h, *wno* assigns a vector in *w* of length n_h that corresponds to the w_{hi} for $0 \le i < n_h$. A similar relationship exists between *obssel* and *obselno*. Selection of observations h is determined by *datasel*, and *betasel* and *betaselno* determine which subvector of the parameter vector $\boldsymbol{\beta}$ applies to h.

The functions irtms and its required functions and the functions that are prerequisites for genrespmle are required by irtmle.

irtms

The function *irtms* finds the log likelihood component $\ell(\boldsymbol{\beta})$ and associated gradient and Hessian matrix for a latent-structure model. As in irtm, the function uses numerical integration if $\mathcal{Y}(A_*, F_*, q_*, r_*)$ is not finite or countably infinite. The function declaration is

f2v irtms (const int & order, const field<pattern> & patterns,
const field<xsel> & patternnumber, const xsel & patno,
const field<field<resp> > & data, const field<field <pwr> > & thetas,
const xsel & thetano,
const field<adq> & scale, const xsel & scaleno, field<dovecmat> & obsscale,
const field<xsel> & selectbeta, const field<xsel> & selectbetano,
const xsel & selbetano,
const field<xsel> & selectbetac, const field<xsel> & selectbetacno,
const xsel & selbetacno,
const field<xsel> & selectthetai, const field<xsel> & selectthetaino,
const xsel & selthetaino,
const field<xsel> & selectthetad, const field<xsel> & selectthetadno,
const xsel & selthetadno,
const field<xsel> & selectthetac, const field<xsel> & selectthetacno,
const xsel & selthetacno,
const field<vec> & w, const xsel & wno, const field<xsel> & obssel,
const xsel & obsselno,
const vec & obsweight, const xsel & datasel,
const field<xsel> & betasel, const xsel & betaselno, const vec & beta).

Definitions are as in *irtmle*, except that *beta* is the general function argument of the log likelihood rather than a starting vector. The function irtm and its required functions are used by irtms.

irtmsave

The function *irtmsave* finds the log likelihood components of $\ell_h(\boldsymbol{\beta})$ and associated gradient and Hessian matrix for a latent structure model. The function declaration is

field<pwrf2v> irtmsave (const int & order, const field<pattern> & patterns,

const xsel & patternnumber, const field<resp> & data, const field <pwr> & thetas, dovecmat & obsscale,
const field<xsel> & selectbeta, const xsel & selectbetano,
const field<xsel> & selectbetac, const xsel & selectbetacno,
const field<xsel> & selectthetai, const xsel & selectthetaino,
const field<xsel> & selectthetad, const xsel & selectthetadno,
const field<xsel> & selectthetac, const xsel & selectthetacno,
const vec & w, const xsel & obssel, const vec & beta).

In this declaration, almost all arguments are defined as in irtm. The struct *pwr* is defined as follows:

struct pwrf2v{double weight; double kernel; resp theta; double value; vec grad; mat hess;}.

For element i of *thetas*, element i of *irtmsave* has *theta* equal to *thetas(i).theta*, *weight* equal to *thetas(i).weight*, *kernel* equal to *thetas(i).kernel*, *value* equal to $\ell_h(\boldsymbol{\beta}|\boldsymbol{\theta})$ for $\boldsymbol{\theta}$ equal to *point* and $\boldsymbol{\beta}$ equal to *beta*, *grad* equal to the corresponding gradient at $\boldsymbol{\beta}$ (if *order* is positive), and *hess* equal to the corresponding Hessian at $\boldsymbol{\beta}$ (if *order* exceeds 1). Functions needed by genresplik are also needed in *irtmsave*.

irtmsaves

The function *irtmsaves* finds the log likelihood components of all the $\ell_h(\boldsymbol{\beta})$ and associated gradient and Hessian matrix for a latent structure model. The function declaration is

field<pwrf2v> irtmsave s(const int & order, const field<pattern> & patterns,
const xsel & patternnumber, const xsel & patno,
const field<field<resp> > & data, const field<field <pwr> > & thetas,
const xsel & thetano,
field<dovecmat> & obsscale,
const field<xsel> & selectbeta, const field<xsel> & selectbetano,
const xsel & selbetano,
const field<xsel> & selectbetac, const field<xsel> & selectbetacno,
const xsel & selbetacno,
const field<xsel> & selectthetai, const field<xsel> & selectthetaino,
const xsel & selthetaino,
const field<xsel> & selectthetad, const field<xsel> & selectthetadno,
const xsel & selthetadno,
const field<xsel> & selectthetac, const field<xsel> & selectthetacno,
const xsel & selthetacno,
const field<vec> & w, const xsel & wno, const field<xsel> & obssel,

const xsel & obsselno,
const vec & obsweight, const xsel & datasel,
const field<xsel> & betasel, const xsel & betaselno, const vec & beta).

Definitions are as in irtms and irtmsave. Element i of *irtmsaves(h)* corresponds to $\ell_h(\boldsymbol{\beta}|\boldsymbol{\theta}_i)$. Functions required are irtmsave and the functions used in irtmsave.

oneparamirt

The main function *oneparamirt* finds maximum-likelihood estimates for a one-parameter item-response model with dichotomous responses and a univariate standard normal latent variable. In this model, a pool of T items numbered from 0 to $T-1$ is used. The model dimension $p = T+1$. Response $\mathbf{Y}_{hi}$ is a one-dimensional vector with the single element Y_{0hi}. The response corresponds to item $t(h,i)$ from the pool, where $t(h,i)$ is a nonnegative integers less than T. Item $t(h,i)$ has possible values 0 and 1, the latent vector $\boldsymbol{\theta}_h$ has dimension 1, and θ_{h0} has a standard normal distribution. The $t(h,i)$ are all different for the same observation h. Conditional on θ_{h0}, Y_{0hi} has a Bernoulli distribution with probability of a value 1 equal to $F(\beta_{t(h,i)} + \beta_T\theta_{h0})$ for a distribution function F in $\mathcal{F}$.

The main function *oneparamirt* considers the simplified case of n_h constant and $t(h,i) = i$ for nonnegative integers $i < n_0$. The function uses a control file *controlfile* read from standard input. The file is a text file with any line consisting of a pair of entries separated by a space, The entries contain no blank characters. The following cases exist.

The variable *data* has string value *infile* that specifies the input file for the data. The default is *infile.csv*. Data are read as an integer matrix with row h and column i equal to Y_{0hi}.

The character variable *dist* specifies the cumulative distribution associated with the binary responses. The possibilities are N for normal, L for logistic, and G for Gumbel. The default is L.

The variable *sf* is associated with the string *startvalue* that specifies a file containing the vector of starting values. The default is to apply startoneparamirt.

The variables *outfile*, *fflag*, *pflag*, *method*, *tol*, *adapt*, i*quadrature*, and *points* are defined as in gpcm.

The program uses irtmle and all its required functions. In addition, startoneparamirt and its required functions, hermpw, qnormpwe, and savmaxf2v are needed.

pcm

The main function *pcm* finds maximum-likelihood estimates for a partial credit model (Masters, 1982) with a standard normal latent variable. The pool of T items and the item responses and latent vectors are defined as in gpcm; however, in the case of the partial credit model, the model dimension p is 1 plus the sum of the $I_t - 1$ for $0 \leq t < T$. In addition, J_{hi0} is now the sum of the $I_u - 1$ for $u < t(h,i)$, and J_{hi1} is the sum of the $I_u - 1$ for $u \leq t(h,i)$. For the nonnegative integer $k < I_{t(h,i)} - 1$, conditional on θ_{h0}, Y_{0hi} has a multinomial logit distribution with element k of the parameter vector $\boldsymbol{\lambda}_{hi}(\boldsymbol{\beta}|\boldsymbol{\theta})$ equal to $\beta_{p-1} + \beta_b k \theta_{h0}$, where $b = J_{hi1}$.

The function *pcm* considers the simplified case of n_h constant and $t(h,i) = i$ for nonnegative integers $i < n_0$ and $I_i - 1$ equal to the maximum value of Y_{0hi} for $0 \leq h < m$. Input is defined as in gpcm.

posterior

For an observation, the function *posterior* finds the posterior distribution of the latent vector given the observed responses. The function declaration is

field<pwr> posterior (const field<pwrf2v> & irtcomps

The definition of *pwr* is in irtm, and the definition of *pwrf2v* is in irtmsave. The posterior is presented as q quadrature weights and points, where *irtcomps* has q members.

posteriors

For a latent-structure model, the function *posteriors* finds the posterior distributions of the latent vectors given the observed responses for each observation. The function declaration is

field<field<pwr> > posteriors (const field<field<pwrf2v> > & irtcompsm

The definition of *pwr* is in irtm, and the definition of *pwrf2v* is in irtmsave. The posterior for each observation h is presented as $q(h)$ quadrature weights and points, where *irtcompsm(h)* has $q(h)$ members.

regprod

The function *regprod* minimizes the sum of squares

$$d(\mathbf{y}, \boldsymbol{\beta}) = \sum_{j=0}^{k-1} [y_{(i(0,j)i(1,j)} - \beta_{i(0,j)}\beta_{i(1,j)}]^2 \tag{29}$$

over β_h, $0 \leq h < p$, where k and p are positive integers, and, for each nonnegative integer $j < k$, $i(0,j)$ and $i(1,j)$ are distinct nonnegative integers less than p and $y_{i(0,j)i(1,j)}$ is a given real number. The function declaration is

maxf2v regprod(const int & order, const params & mparams, const char & algorithm, const field<resp> & y, const vec & start), where *maxf2v* and the initial three function arguments are defined as in irtmle, y has p members, *y(j).iresp* has elements $i(0,j)$ and $i(1,j)$, and *y(j).dresp* has the single element $y_{i(0,j)i(1,j)}$. In the selected algorithm, the starting values for the β_h are provided by the p elements of *start*.

The algorithm uses regprodf together with maxselect and its required functions.

regprodf

The function *regprodf* computes Equation 29. The function declaration is

f2v regprodf(const int & order, const field<resp> & y, const vec & beta).

The function arguments are defined as in regprod.

startgpcm

The function *startgpcm* finds starting values for gpcm. The function declaration is

vec startgpcm(const int & order , const params & mparams, const char & algorithm, const imat & responses)

The first three arguments are defined as in irtmle, and *responses* is the n by m matrix of responses. Required functions are regprod and its required functions.

startoneparamirt

The function *startoneparamirt* finds startig values for oneparamirt. The function declaration is

vec startoneparamirt(const char & cdf, const imat & responses).

The variable *cdf* is defined as in berresp, and *responses* is defined as in startgpcm. The required function is invcdf.

startpcm

The function *startpcm* finds startig values for pcm. The function arguments are

vec startpcm(const imat & responses).

The function argument *response* is defined as in startgpcm.

starttwoparamirt

The function *starttwoparamirt* finds a starting vector for use in twoparamirt. The function declaration is

vec starttwoparamirt(const int & order , const params & mparams, const char & algorithm, const char & cdf, const imat & responses).

Arguments are defined as in startgpcm and startoneparamirt. The function regprod and its required functions are needed.

twoparamirt

The program *twoparamirt* provides a basic analysis of a two-parameter item-response model for binary responses with a univariate standard normal latent variable. For each observation, the same items apply. It uses a control file *controlfile* read from standard input. The file is a text file with any line consisting of a pair of entries separated by a space, The entries contain no blank characters. The following cases exist.

The variable *data* has string value *infile* that specifies the input file for the data. The default is *infile.csv*. Each line of the input file represents a different observation and contains the same number of entries, with each entry the response to an item. Items are presented in the same order for each observation.

The variable *sf* is associated with the string *startvalue* that specifies a file containing the vector of starting values. The default is to apply starttwoparamirt.

The variable *outfile* has string value that specifies the name of the output file for results. The default is *outfile*.

The bool variable *fflag* indicates whether an output file is used. The default is *true*.

The bool variable *pflag* indicates whether anything is printed in ascii form in standard output. The default is *true*.

The character variable *dist* specifies the cumulative distribution associated with the binary responses. The possibilities are *N* for normal, *L* for logistic, and *G* for Gumbel. The default is *L*.

The character variable *method* specifies the algorithm applied. The possibilities are *G* for gradient ascent, *C* for conjugate gradient ascent, *N* for modified Newton-Raphson, and *L* for Louis approximation. The default is *N*.

The positive double variable *tol* specifies the convergence criterion *mparams.tol*. The default is 0.001.

The bool variable *adapt* indicates whether adaptive quadrature is used. The default is *true*.

The character variable *quadrature* indicates the quadrature procedure used, with *G* for Gauss-Hermite and *Q* for normal quantiles. The default is *G*.

The positive integer variable *points* that exceeds 1 gives the number of quadrature points, with the default 9.

The program uses irtmle and all its required functions. In addition, starttwoparamirt and its required functions, hermpw, qnormpwe, and savmaxf2v are needed.

Integration Tools

eaps

For each observation *i*, the function *eaps* generates a posterior weighted mean *eaps(i).v* and covariance matrix *eaps(i).m* that correspond to a discrete posterior distribution with points *posts(i).m.col(j)* with probabilities *posts(i).v(j)* for *j* from 1 to *posts(i).m.n_cols*. The function declaration is
field<vecmat> eaps(const field<vecmat> & posts).
The struct *vecmat* has the form

struct vecmatvec v; mat m.

The function wmc is used.

fitquad

The function *fitquad* fits a quadratic function to function values and quadrature points and finds the linear transformation that maps the origin onto the location of the maximum of the quadratic function and has a symmetric Jacobian matrix equal to the positive-definite and symmetric square root of the inverse of the Hessian matrix of the fitted quadratic function. The function declaration is

dovecmat fitquad(const field<f2v> & cresults, const field<pwr> & newthetas,
const adq & scale, dovecmat & obsscale).

The structs are defined as in irtm. The linear components of the fitted quadratic not set to 0 are specified by *scale.linselect*, and the quadratic components not set to 0 are specified by *scale.quadselect*. If the fitted quadratic function is not strictly concave, then *fitquad* is *obsscale*.

genfact

For a vector *sizes* of positive integers, the function *genfact* generates all vectors *i* of nonnegative integers with the same number of elements as *sizes* such that each element of *i* is less than the corresponding element of *sizes*. The function declaration is

imat genfact(const ivec & sizes).

The columns of *genfact* are the possible vectors *i*. For example, if the elements of *sizes* are 2 and 3, then Column 0 of *genfact* has elements 0 and 0, and Column 1 has elements 1 and 0. In all, *sizes* has 6 columns, and Column 5 has elements 1 and 2.

genprods

The function genprods generates a collection of quadrature points and quadrature weights for a multivariate integral from quadrature weights and quadrature points for a univariate integral. The function declaration is

genprods(const imat & indices, const field<pw> & pws).

The struct *vecmat* is defined as in eaps, and the struct *pw* has *vec* elements *points* and *weights*. Consider the case of Q quadrature points for a multidimensional integral on the space of D-dimensional vectors, where Q and D are positive integers. Then *genprods.m* has Q columns and *genprods.v* has Q elements. The matrix *genprods.m* has D rows. The array *pws* has D members. For $0 \leq d < D$, *pws(d).points* and *pws(d).weights* have $m(d) > 1$ members, and the members of *pws(d).weights* are positive. The matrix *indices* specifies the quadrature vectors and quadrature weights to construct from *pws*. If *indices* has p columns, $0 \leq k < p$, and $0 \leq d < D$, then row d and column k of *indices* is nonnegative and less than $m(d)$ and the corresponding row and column of *genprods.m* is *pws(d).points(indices(d,k))*. Element k of *genprods.v* is the product of *pws(d).weights(indices(d,k))* for $0 \leq d < D$.

hermcoeff

The function hermcoeff finds the coefficients of a Hermite polynomial of a given degree. The function declaration is

vec hermcoeff(const int & n).

The integer variable *n* is the nonnegative order. The vector *hermcoeff* has *n*+1 elements. The polynomial is $H_n(x) = \sum_{i=0}^{n} \alpha_i x^{n-i}$ for real x, and element i of *hermcoeff* is α_i. For example, if *n* is 2, then the elements of *hermcoeff* are 1, 0, and -1.

hermpoly

The function hermpoly evaluates the Hermite polynomials up to a given degree at a specified real value. The function declaration is

vec hermpoly(const int &n, const double & x).

The degree is the nonnegative integer variable n, and the real value is x. The vector *hermpoly* has n+1 elements. For $0 \leq k \leq n$, element k of *hermpoly* is the value of H_k at x.

hermpw

The function hermpw uses the algorithm of Golub and Welsch (1969) to find the quadrature points and quadrature weights for Gauss-Hermite quadrature. The function declaration is

pw hermpw(const int & n).

The struct *hermpw* has vector elements *hermpw.points* and *hermpw.weights*. The number of quadrature points is *n*. The ordered quadrature points are in *hermpw.points*. The corresponding weights are in *hermpw.weights*. The weights are relative to the standard normal density.

qnormpwe

The function *qnormpwe* provides normal-scores quadrature of a given order. The function declaration is

pw qnormpwe(const int & n).

The struct *qnormpwe* has vector elements *qnormpwe.points* and *qnorm-pwe.weights*. The number of quadrature points is *n*. The ordered quadrature points are in *qnormpwe.points*. The corresponding weights are in *qnormpwe.weights*.

wmc

The function wmc computes a weighted mean vector and covariance matrix. The function declaration is

vecmat wmc(const vecmat & wx).

The struct *vecmat* is defined as in eaps. The elements of *wx.v* are probabilities corresponding to the rows of *wx.m*.

References

Anderson, T. W. (2003). *An introduction to multivariate statistical analysis* (3rd ed.). Wiley-Interscience.

Cox, D. R. (1972). Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, *34*, 187–202. https://doi.org/10.1111/j.2517-6161.1972.tb00899.x.

Golub, G. H., & Welsch, J. H. (1969). Calculation of gauss quadrature rules. *Mathematics of Computation*, *23*, 221–s10. https://doi.org/10.2307/2004418.

Gumbel, E. J. (1935). Les valeurs extrémes des distributions statistiques. *Annales de l'Institut Henri Poincaré*, *5*, 115–158.

Haberman, S. J. (1974). *The analysis of frequency data*. University of Chicago Press.

Haberman, S. J. (1977). Maximum likelihood estimates in exponential response models. *The Annals of Statistics*, *5*, 815–841. https://doi.org/10.1214/aos/1176343941.

Haberman, S. J. (1980). Discussion of *regression models for ordinal data*, by P. McCullagh. *Journal of the Royal Statistical Society, Series B*, *42*, 136–137.

Haberman, S. J. (2013). *A general program for item-response analysis that employs the stabilized Newton-Raphson algorithm* (ETS Research Report No. RR-13-32). Educational Testing Service. https://doi.org/10.1002/j.2333-8504.2013.tb02339.x.

Kalbfleisch, J. D., & Prentice, R. L. (2002). *The statistical analysis of failure time data* (2nd ed.). John Wiley. https://doi.org/10.1002/9781118032985.

Lord, F. M., & Wingersky, M. S. (1984). Comparison of IRT true-score and equipercentile observed-score "equatings". *Applied Psychological Measurement*, *8*, 453–461. https://doi.org/10.1177/014662168400800409.

Louis, T. (1982). Finding the observed information matrix when using the *em* algorithm. *Journal of the Royal Statistical Society, Ser. B*, *44*, 226–233. https://doi.org/10.2307/2345828.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika*, *47*, 149–174. https://doi.org/10.1007/bf02296272.

McCullagh, P., & Nelder, J. A. (1989). *Generalized linear models* (2nd ed.). Springer US. https://doi.org/10.1007/978-1-4899-3242-6.

McFadden, D. L. (1973). Conditional logit analysis of qualitative choice behavior. In P. Zarembka (Ed.), *Frontiers in econometrics* (pp. 105–142). Academic Press.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement*, *16*, 159–176. https://doi.org/10.1177/014662169201600206.

Muthén, B. O. (2002). Beyond SEM: General latent variable modeling. *Behaviormetrika*, *29*, 81–117. https://doi.org/10.2333/bhmk.29.81.

Sanderson, C., & Curtin, R. (2016). Armadillo: A template-based C++ library for linear algebra. *The Journal of Open Source Software*, *1*, 26. https://doi.org/10.21105/joss.00026.

Sanderson, C., & Curtin, R. (2018). A user-friendly hybrid sparse matrix class in C++. In *Mathematical software – ICMS 2018* (pp. 422–430). https://doi.org/10.1007/978-3-319-96418-8_50.

Thissen, D., Pommerich, M., Billeaud, K., & Williams, V. S. L. (1995). Item response theory for scores on tests including polytomous items with ordered responses. *Applied Psychological Measurement*, *19*, 39–49. https://doi.org/10.1177/014662169501900105.

van der Linden, W. J. (Ed.). (2018). *Handbook of item response theory, three volume set*. Chapman; Hall/CRC. https://doi.org/10.1201/9781315119144.