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
- conjgrad
- contresp
- cumresp
- cx
- eaps

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.

- fitquad
- genfact
- genprods
- genresp
- genresplik
- genrespmle
- gradascent
- gradresp
- gumbel
- hermcoeff
- hermpoly
- hermpw
- intsel
- invcdf
- irtm
- irtmle
- irtms
- irtmsave
- irtmsaves
- ivecsel
- linsel
- loggamma
- logistic
- logit
- logitbeta

- logitdirichlet
- loglog
- logmean
- lw
- lwm
- maxberresp
- maxf2vvar
- maxlinq2
- maxquad
- maxselect
- multlogit
- modit
- normal
- normalv
- normwt
- nrv
- pack
- posterior
- posteriors
- probit
- qnormpw
- ranklogit
- rebound
- savmaxf2v
- sintsel

- sivecsel
- starttwoparamirt
- svecsel
- truncresp
- twoparamirt
- unpack
- 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).

The vector p has dimension n and has positive elements that are less than 1. For $0 \leq j < n$, the probability that $X_j = 1$ is element j of p. 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 that $S = k$.

lwm

The function *lwm* finds the probability mass function of the sum S of n mutually independent random variables X_j, $0 \leq j <$ 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 members. For $0 \leq j < n$, member j of p is the vector *p[j]* with I_j nonnegative elements. The sum of these elements is 1, and element k, $0 \leq k < I_j$, of *p[j]* is the probability that $X_j = k$. The probability mass function is provided by *lwm*, a vector with $K = 1 + \sum_{j=1}^{n}(I_j - 1)$ elements. Element k of *lwm*, $0 \le k < K$, is the probability that $S = k$. 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 algorithm used by lwm reduces to the conventional generalization of the Lord-Wingersky algorithm to sums of independent multinomial variables.

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.value* 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.value* is at least a is closed and bounded, and the sets A_0 of members of O at which *f.value* exceeds a is nonempty. The function *f.value* 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.value* has value *NaN* at any p-dimensional vector 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;};,

vary0.locmax is the starting vector for the line search, *vary0.max* is the value of *f.value* at the starting vector, *maxlinq2.grad* is the gradient of *f.value* at *vary0.locmax*, *maxlinq2.hess*, if computed, is the gradient of *f.value* at *vary0.locmax*, and *maxlinq.locmax* is the approximate location of the maximum of *f.value* on the half-line that starts at *vary0.locmax* and has direction *v*, *maxlinq2.max* is the approximate maximum of *f.value* on the half-line, *maxlinq2.grad* is the gradient of *f.value* at *maxlinq.locmax*, and *maxlinq2.hess*, if computed, is the Hessian of *f.value* at *vary0.locmax*,

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.

The definition of *f2v* is

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

where *f.value* is the function value, *f.grad* is the gradient of *f.value*, and *f.hess* is the Hessian of *f.value*.

The functions maxf2vvar, maxquad, modit, and rebound are all used.

maxquad

The function *maxquad* approximates the maximum of *f.value* along a half-line by use of a quadratic two-point approximation. 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* and *x1* are the points used, *f0* is the function value at *x0*, *f1* is the function value at *x1*, *g0* is the derivative at *x0*, and *stepmax* is the maximum change from *x0* permitted in the estimated location *maxquad* of the function maximum.

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, *alpha1* is the proposed new location, *stepmax* is the positive limit on step size, *b.lower* is the lower bound, and *b.upper* is the upper bound. It is assumed that *alpha0* and *alpha1* are different. The function returns a value *modit* that is normally *alpha1*; however, if *alpha1* exceeds *alpha0*, then *modit* is truncated above so that it does not exceed the minimum of *alpha0+stepmax* and *alpha0+eta(b.upper-alpha0)*, while if *alpha1* is less than *alpha0*, then *modit* is truncated below so that it is at least the maximum of *alpha0-stepmax* and *alpha0+eta(b.lower-alpha0)*.

rebound

The function *rebound* updates the lower and upper bounds for maximization of a differentiable real function 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, *der* is the function derivative at *y*, *b.lower* is the current lower bound, and *b.upper* is the current upper bound. It is assumed that *der* is not 0. If *der* is positive, *modit.lower* is *y* and *modit.upper* is *b.upper*. If *der* is negative, *modit.upper* is *y* and *modit.lower* is *b.lower*.

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.value* 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. The returned value *maxf2vvar.locmax* is *y*, while *maxf2vvar.max* is *fy.value*, *maxf2var.grad* is *fy.grad*, and *maxf2var.hess* is *fy.hess* at *y*. If *order* is less than 1, only fy.value is considered If *order* is 1, *fy.value* and *fy.grad* are considered. If *order* exceeds 1, then *fy.value*, *fy.grad*, and *fy.hess* are used.

nrv

The function *nrv* applies a modified version of the Newton-Raphson algorithm to maximization of *f.value*. 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 starting vector *start* must be in O.

The function *nrv* uses maxf2vvar, maxlinq2, maxquad, modit, and rebound. The value of *order* should be at least 2.

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 *f.value*. The function declaration is

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

The starting vector is *start*. 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.value*. 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.value*. 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 $\ell_c(\boldsymbol{\beta};\mathbf{Y},A,F,q,r)$. 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), D (continuous), E (logit Dirichlet), G (graded), H (log gamma), L (multinomial logit), 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 three members, G, the standard Gumbel distribution function with value

$G(y) = 1 - \exp(-\exp(-y))$ 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 and a set $\mathcal{Y}(A, F, q, r)$ of r-dimensional vectors. The vector $\boldsymbol{\beta}$ is in $O(A, F, q, r)$, and $\mathbf{Y}$ is in $\mathcal{Y}(A, F, q, r)$.

To treat both continuous and discrete log-likelihood components, the integral symbol $\int$ is used in the following sense. Consider a real function g on a nonempty finite-dimensional set C. If C is convex and has a nonempty interior and g is integrable, then $\int(g)$ denotes the integral of g over C. If C is finite or countably infinite and g is summable, then $\int(g)$ is the sum of $g(\mathbf{c})$ over $\mathbf{c}$ in C. More generally, let $\mathcal{D}$ be a finite or countably infinite collection of nonempty disjoint sets D that are either convex sets with nonempty interior or finite or countably-infinite sets. Let C be the union of the sets in $\mathcal{D}$, and let g_D denotes the restriction of g to D in $\mathcal{D}$. Let $\int(g_D)$ be defined for D in $\mathcal{D}$, and let the $\int(g_D)$, D in $\mathcal{D}$, be summable. Then $\int(g)$ is the sum of $\int(g_D)$ over D in $\mathcal{D}$. Similar conventions apply if g is vector-valued or matrix-valued. The requirement is imposed here that, for $\boldsymbol{\beta}$ in $O(A, F, q, r)$, $\int(\exp(\ell_c(\boldsymbol{\beta}; \cdot, A, F, q, r))) = 1$. Here $\exp(\ell_c(\boldsymbol{\beta}; \cdot, A, F, q, r))$ is the function on $\mathcal{Y}(A, F, q, r)$ equal to $\exp(\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r))$ if $\mathbf{y}$ is in $\mathcal{Y}(A, F, q, r)$. The gradient function of $\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ is $\nabla\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ and the corresponding Hessian matrix is $\nabla^2\ell_c(\cdot; \mathbf{y}, A, F, q, r)$.

For a positive integer n and an observation i, $0 \le i < n$, 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 $\boldsymbol{\tau}$ is in O, then let $\boldsymbol{\lambda}_i(\boldsymbol{\tau}) = \mathbf{o}_i + \mathbf{X}_i\boldsymbol{\tau}$ 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

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

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

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

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

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

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

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

(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_i, then all values returned equal *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 $\hat{\boldsymbol{\tau}}$ in O such that $\ell(\hat{\boldsymbol{\tau}})$ equals the supremum of ℓ over O are relatively complex (Haberman, 1974, 1977, 1980). It is worth noting that in cases in which $\hat{\boldsymbol{\tau}}$ in O satisfies the conditions that $\nabla\ell(\hat{\boldsymbol{\tau}})$ is the p-dimensional vector $\mathbf{0}_p$ with all elements 0 and $\nabla^2\ell(\hat{\boldsymbol{\tau}})$ is negative definite, then O can be restricted to ensure that ℓ is strictly concave on O and $\hat{\boldsymbol{\tau}}$ is the only member of O such that $\ell(\hat{\boldsymbol{\tau}})$ equals the supremum of ℓ on O and, for $\boldsymbol{\tau}$ in O, $\nabla\ell(\boldsymbol{\tau})$ is only the 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 *vec* component *dresp* and *ivec* component *iresp*. 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 $y(0)$ 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 $O(A,F,q,r)$,

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

The function declaration is

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

If *transform* is G, then $F = G$, 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 loglog, logit, and probit.

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 of all two-dimensional vectors $\boldsymbol{\beta}$ with element $\beta(1) > 0$, and F is in $\mathcal{F}$. For $\mathbf{y}$ in $\mathcal{Y}(A, F, q, r)$ and $\boldsymbol{\beta}$ in $O(A, F, q, r)$,

$$\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r) = \log(\beta(1)) + \log(F_1(\beta(0) + \beta(1)y(0))). \tag{6}$$

These cases correspond to a model that a random variable Y has a distribution function $F(\beta(0) + \beta(1)y)$, where F is the distribution function of a random variable Z. Here $\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ is concave, and the function is strictly concave if $y(0)$ is not 0.

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 gumbel, logistic, and normal.

cumresp

The function *cumresp* computes the function value, gradient, and Hessian matrix associated with a cumulative response transformation. 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 $\mathcal{Y}(A, F, q, r)$,

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

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, loglog, 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.

gradresp

The function *gradresp* computes the function value, gradient, and Hessian matrix associated with a graded response transformation. 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 $y(0)$ a nonnegative integer no greater than q, and, for $\boldsymbol{\beta}$ in $O(A, F, q, r)$ and $\mathbf{y}$ in $\mathcal{Y}(A, F, q, r)$,

$$\ell_i(\boldsymbol{\beta}; \mathbf{y}) = \begin{cases} \log(1 - F(\beta(y(0))), & y(0) = 0, \\ \log(F(\beta(y(0) - 1)) - F(\beta(y(0)))), & 0 < y(0) < q, \\ \log(F(\beta(y(0) - 1))), & y(0) = q. \end{cases} \tag{8}$$

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. The function $\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ is concave. Strict concavity only holds if q is 1 or q is 2 and $y(0) = 1$.

gumbel

The function *gumbel* provides the computations required in contresp for $\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ for the simple Gumbel case of $F = G$, 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 gumbel(const int & order, const resp & y, const vec & beta).

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

loggamma

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

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

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

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 $\boldsymbol{\beta}$.

logitbeta

The function *logitbeta* computes the function value, gradient, and Hessian matrix associated with the logit of a beta distribution with a two-dimensional parameter vector $\boldsymbol{\beta}$ with positive elements. Here $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ has A with value H, $q=2$, $r=1$, $\mathcal{Y}(A,F,q,r)$ the set of real numbers, and $O(A,F,q,r)$ the set of two-dimensional vectors $\boldsymbol{\beta}$ with positive elements. The function declaration is

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

The function *logitbeta.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if $q=2$, $r=1$, *y.iresp* is $\mathbf{y}$, A is B, and *beta* is $\boldsymbol{\beta}$.

logitdirichlet

The function *logitdirichlet* computes the function value, gradient, and Hessian matrix associated with the logits of a Dirichlet distribution with a $q=r+1$-dimensional parameter vector $\boldsymbol{\beta}$ with positive elements. 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 r-dimensional vectors, and $O(A,F,q,r)$ the set of q-dimensional vectors $\boldsymbol{\beta}$ with positive elements. The function declaration is

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

The function *logitdirichlet.value* is $\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)$ if *y.iresp* is $\mathbf{y}$, *beta* is $\boldsymbol{\beta}$, and A is E. The q-dimensional random variable $\mathbf{u}$ has a Dirichlet distribution with parameter vector $\boldsymbol{\beta}$ if 1 is the sum of the elements of $\mathbf{u}$ and $y_i = \log(u_i/u_q)$ for integers i from 1 to r. If $r = 1$, then the logit Dirichlet case reduces to the case of a logit beta.

loglog

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

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

The function *loglog.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 $\mathcal{Y}(A,F,q,r)$,

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r) = y(0)\beta(0) - \exp(\beta(0)) - \log([y(0)]!). \tag{9}$$

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 $O(A,F,q,r)$,

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r) = \begin{cases} \log(F(\beta(0) + F(\beta(1) - F(\beta(0)F(\beta(1)), & y(0) = 1, \\ \log(1 - F(\beta(0))) + \log(1 - F(\beta(1))), & y(0) = 0. \end{cases} \tag{10}$$

It should be noted that

$$F(\beta(0) + F(\beta(1) - F(\beta(0)F(\beta(1)) = 1 - [1 - F(\beta(0))][1 - F(\beta(1)] \tag{11}$$

and

$$\log(1-F(\beta(0))+\log(1-F(\beta(1))=\log([1-F(\beta(0)][1-F(\beta(1)]). \qquad (12)$$

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, loglog, 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 $\mathcal{Y}(A,F,q,r)$,

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)=\begin{cases} -\log\left(1+\sum_{k=0}^{q-1}\exp(\beta(k)\right), & y(0)=0,\\ \beta(y(0)-1)+\ell_c(\boldsymbol{\beta};\mathbf{0}_1,A,F,q,r), & y(0)>0.\end{cases} \qquad (13)$$

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 $\boldsymbol{\beta}$ with $\beta(1)>0$. 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 $\boldsymbol{\beta}$.

normalv

The function normalv computes the function value, gradient, and Hessian matrix associated with the log-likelihood component associated with a multivariate normal model with r positive, $q = r(r+3)/2$, A equal to N, F irrelevant, $\mathcal{Y}(A, F, q, r)$ the set of all r-dimensional real vectors, and $O(A, F, q, r)$ the set of q-dimensional vectors $\boldsymbol{\beta}$ with elements β_h, $0 \le h < q$ such that $\beta_h > 0$ if $h = r + j(j+3)/2$ and $0 \le j < r$. For such $\boldsymbol{\beta}$, let $\mathbf{a}(\boldsymbol{\beta})$ be the r-dimensional vector with elements $a_j(\boldsymbol{\beta}) = \beta_j$ for $0 \le j < r$, and let $\mathbf{B}(\boldsymbol{\beta})$ be the symmetric positive-definite r by r matrx with row j and column k equal to β_h if $0 \le k \le j < r$ and $h = r + k + (j(j+1)/2$. For an r-dimensional vector $\mathbf{z}$ with elements z_j, $0 \le j < q$, let $\phi(\mathbf{z}; r)$ be the product of the $\Phi_1(z_j)$, $0 \le j < r$.

For $\mathbf{y}$ in $\mathcal{Y}(A, F, q, r)$,

$$\ell_c(\boldsymbol{\beta}; \mathbf{y}) = \left[\sum_{j=0}^{r-1} \log(\beta(r + j(j+3)/2))\right] + \log(\phi(\mathbf{a}(\boldsymbol{\beta}) + \mathbf{B}(\boldsymbol{\beta})\mathbf{y}; r)). \tag{14}$$

This case corresponds to a model that a random vector has a distribution $\mathbf{a}(\boldsymbol{\beta}) + \mathbf{B}(\boldsymbol{\beta})\mathbf{Z}$, where $\mathbf{Z}$ is an r-dimensional multivariate normal random vector with zero mean and with covariance matrix equal to the identity matrix. The function $\ell_c(\cdot; \mathbf{y}, A, F, q, r)$ is always concave but is not strictly concave. The function declaration is

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

The function *normalv.value* is $\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r)$ if *y.dresp* is $\mathbf{y}$ and *beta* is $\boldsymbol{\beta}$. If r is 1, then *normalv* reduces to normal. The function *normalv* requires pack and unpack.

pack

For the struct *vecmat* defined by

struct vecmax{vec v; mat m;};,

the function *pack* converts a d-dimensional vector *pack.v* and a d by d symmetric matrix *pack.m* to a vector with dimension $d(d+3)/2$ with d initial elements the vector *pack.v* and element $h = d + k + (j(j+1)/2$ equal to row j and column k of *pack.m* for nonnegative k no greater than $j < d$. The function declaration is

vec pack(const vecmat & u).

Diagonal elements of the matrix equal the corresponding elements of u, and off-diagonal elements are twice the corresponding elements of the vector.

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 $r\le q$. 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 the standard Gumbel distribution function G. Consider $\boldsymbol{\beta}$ in $O(A,F,q,r)$. Let $U(j)$, $0\le j\le q$, be independent random variables such that $U(0)$ and $U(j)-\beta(j)$, $1\le j\le q$, have the common distribution function G. Let $\mathbf{Y}$ be a random vector with values in $\mathcal{Y}(A,F,q,r)$ such that $\mathbf{Y}$ is the member $\mathbf{y}$ of $\mathcal{Y}(A,F,q,r)$ with elements $y(j)$, $0\le j<r$, if $U(y(j))$ is nonincreasing in j and $U(y(j))\ge U(k)$ if k is a nonnegative integer no greater than q that does not equal $y(h)$ for any nonnegative integer element $h<r$. For $\boldsymbol{\beta}$ in $O(A,F,q,r)$ and $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$, let $\boldsymbol{\alpha}(\boldsymbol{\beta})$ be the vector of dimension $q+1$ such that element j, $0\le j\le q$, is $\alpha(j;\boldsymbol{\beta})=0$ if $j=0$ and $\alpha(j;\boldsymbol{\beta})=\beta(j-1)$ if $j>0$. For $\mathbf{y}$ in $\mathcal{Y}(A,F,q,r)$ and $0\le j<r$, let $K(j;\mathbf{y})$ be the set of nonnegative integers no greater than q not equal to $y(h)$ for any nonnegative integer $h<j$. Thus $K(0;\mathbf{y})$ is the set of nonnegative integers no greater than q. Then the log-likelihood component is

$$\ell_c(\boldsymbol{\beta};\mathbf{y},A,F,q,r)=\sum_{j=0}^{r-1}\left[\alpha(y_j;\boldsymbol{\beta})-\log\left(\sum_{h\in K(j;\mathbf{y})}\exp(\alpha(h;\boldsymbol{\beta}))\right)\right]. \tag{15}$$

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 $\beta(0)$ and positive real $\beta(1)$, 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

$$\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r) = \log(\beta(1)) + \log(F_1(\beta(0) + \beta(1)y(0))), \tag{16}$$

while in the case of $y(1) = 1$, the the observation is censored at $y(0)$ and the log-likelihood component is

$$\ell_c(\boldsymbol{\beta}; \mathbf{y}, A, F, q, r) = \log(1 - F(\beta(0) + \beta(1)y(0))). \tag{17}$$

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.

unpack

The function *unpack* converts a vector of dimension $d(d+3)/2$ to the *vecmat* format described in pack. The function declaration is

vecmat unpack(const int & d, const vec & beta).

The vector *unpack.v* contains the first d elements of *beta* and row j and column k of *unpack.m* is element $d + k + j(j + 3)/2$ of *beta* for nonnegative integers $k \le j < d$.

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 logit beta, C for a cumulative case, D for a continuous case, E for logit Dirichlet, G for a graded response, H for log gamma, L for the multinomial logit case, M for the maximum of two independent Bernoulli variables, N for the multivariate normal case, P for the log-mean Poisson case, R for the rank-logit case, S for the Bernouli case, and T for the censored continuous case. For discrete cases, *choice.transform* has possible values G for log-log cases, L for logit cases, and N for probit cases. For continuous cases, G is for the Gumbel distribution, L is for the logistic case, and N is for the normal case. For example, *choice.type* is C and *choice.transform* is G for the cumulative 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, loggamma, logitbeta, logitdirichlet, 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 resp & theta,
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).

The definitions of arguments and structs rely the the definition of *model* in genresp and the definition of *resp* in truncresp. The variable *theta* is a supplemental variable often used with item-response models.

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 in the case of *obssel*, the observation numbers i to be used are specified. The function intsel is used for selections based on *xsel*. If *xsel.bool* is *true*, then an integer i is mapped by intsel to i, whereis if *bool* is *false*, then i is mapped to element i of *xsel.list*.

If *patternnumber* maps the integer i to $j(1)$, then *patterns(j(1))* 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*, and *selectbeta(j(2)* specifies the elements of *beta* used to define $\boldsymbol{\lambda}_i$ if *selectbetano* maps i to $j(2)$.

The struct *pattern* is defined as

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

Thus *patterns(j(1)).choice* defines the model for $\mathbf{Y}_i$. The vector *patterns(j(1)).o* is then $\mathbf{o}_i$, while *patterns(j(1)).x* and *patterns(j(1)).c* are used to find $\mathbf{X}_i$. Definition of $\mathbf{X}_i$ is somewhat complex. Let *selectbetacno* map i to $j(2)$, and let *selectthetacno* map i to $j(3)$. If *k* is not selected by *selectbeta(j(2))*, then column *k* of $\mathbf{X}_i$ is the zero vector. Otherwise, let $k(2)$ be element $m(2)$ selected by *selectbeta(j(2))*. If $m(2)$ is not selected by *selectbetac(j(2))*, then column $k(2)$ of $\mathbf{X}_i$ is column $m(2)$ of *patterns(j(1)).x*. Otherwise, let $m(2)$ be element $m(3)$ selected by *selectbetac(j(2))*. Then column $k(2)$ of $\mathbf{X}_i$ is the sum of column $m(2)$ of *patterns(j(1)).x* and column $m(3)$ of the sums of the products of *theta.dresp(k(4))* times slice $m(4)$ of *patterns(j(1)).c* for pairs $k(4)$ and $m(4)$ such that $k(4)$ is integer $m(4)$ selected by *selectthetac(j(3))*.

In the case of $\mathbf{Y}_i$, let *thetaino(i)* select $j(5)$ and *thetadno(i)* select $j(6)$. If *thetai(j(5))* does not select anything, *data(i).iresp* is the int component of $\mathbf{Y}_i$. Otherwise, the int component of $\mathbf{Y}_i$ consists of the elements of *theta.iresp* selected in *thetai(j(5))*. If *thetad(j(6))* does not select anything, *data(i).dresp* is the double component of $\mathbf{Y}_i$. Otherwise, the double component of $\mathbf{Y}_i$ consists of the elements of *theta.dresp* selected in *thetad(j(6))*. The function *genresplik* uses addsel, cx, intsel, linsel, sintsel, sivecsel, svecsel, vecsel, genresp, and all C++ functions genresp requires.

genrespmle

The function *genrespmle* applies maximizes the log-likelihood 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 resp & theta,
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 & 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.

cx

The function *cx* is used to multiply a cube by a vector to yield a matrix. The function declaration is

mat cx(const cube & c, const vec & x).

The result is that *cx(i,j)* is the sum over k of $c(i,j,k)x(k)$.

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 from *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 *xselect.list*, and element i of *ivecsel* is element *xselect.list(i)* of *y*.

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. It is always the case that *linsel.value* is *x.value*. If *order* is positive, then *linsel.grad* is the product of the transpose of *a* and the gradient *x.grad*. If *order* exceeds 1, then *linsel.hess* is the product of the transpose of *a*, the Hessian *x.hess*, and the matrix *a*.

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 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 integer 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 *xselect.list(i)* of *y*.

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, use of latent variables is involved. In typical cases, data involve multiple responses for each individual observation. For a positive integer m, m observations are present. For observation h, $0 \le h < m$, the observation has weight $w_{h*} > 0$, and n_h responses are observed. In addition, a latent vector appears in the model. 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 $\mathcal{Y}(A_*, F_*, q_*, r_*)$. The latent variable 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 $\boldsymbol{\lambda}_*(\boldsymbol{\tau}) = \mathbf{o}_* + \mathbf{X}_*\boldsymbol{\tau}$ is in $O(A_*, F_*, q_*, r_*)$ as long as $\mathbf{X}_*$ is in $\mathcal{X}_*$ and $\boldsymbol{\tau}$ 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 $\boldsymbol{\tau}$ in O, $\mathbf{X}$ in $\mathcal{X}_{hi}$, and $\boldsymbol{\theta}$ in $\mathcal{Y}(A_*, F_*, q_*, r_*)$,

$$\boldsymbol{\lambda}_{hi}(\boldsymbol{\tau}|\boldsymbol{\theta}) = \mathbf{o}_{hi} + \mathbf{X}_{hi}\boldsymbol{\tau} + \mathbf{D}_{hi}\boldsymbol{\theta} + \sum_{k=0}^{p-1} \theta_k \mathbf{D}_{hik}\boldsymbol{\tau} \tag{18}$$

is in $O(A_{hi}, F_{hi}, q_{hi}, r_{hi})$.

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

$$\ell(\boldsymbol{\tau}) = \sum_{h=0}^{m-1} w_{h*}\ell_h(\boldsymbol{\tau}), \tag{19}$$

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

$$\nabla\ell(\boldsymbol{\tau}) = \sum_{h=0}^{m-1} w_{h*}\nabla\ell_h(\boldsymbol{\tau}), \tag{20}$$

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

$$\nabla^2\ell(\boldsymbol{\tau}) = \sum_{h=0}^{m-1} w_{h*}\nabla^2\ell_h(\boldsymbol{\tau}), \tag{21}$$

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

$$\tilde{\nabla}^2\ell(\boldsymbol{\tau}) = -\sum_{h=0}^{m-1} w_{h*}\nabla\ell_h(\boldsymbol{\tau})[\nabla\ell_h(\boldsymbol{\tau})]^T, \tag{22}$$

may also be considered.

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

$$\ell_h(\boldsymbol{\tau}|\boldsymbol{\theta}) = \ell_c(\boldsymbol{\lambda}_*(\boldsymbol{\tau}); \boldsymbol{\theta}, A_*, F_*, q_*, r_*) \sum_{i=0}^{n_h-1} w_{hi}\ell_c(\boldsymbol{\lambda}_{hi}(\boldsymbol{\tau}|\boldsymbol{\theta}); \mathbf{Y}_{hi}, A_{hi}, F_{hi}, q_{hi}, r_{hi}) \tag{23}$$

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

$$\ell_h(\boldsymbol{\tau}) = \log\int(\exp(\ell_h(\boldsymbol{\tau}|\cdot))), \tag{24}$$

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

$$\tilde{\ell}_h(\boldsymbol{\tau}) = \log\left[\sum_{k=1}^{Q} u_{hk}\exp(\ell_h(\boldsymbol{\tau}|\boldsymbol{\theta}_{hk}))\right], \tag{25}$$

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.

invcdf

The function *invcdf* finds the inverse of the cumulative distribution function and its derivative at *prob* for either the standard Gumbel (*cdf* equals G), logistic (*cdf* equals L), or normal (*cdf* equals N). The function declaration is

f1 invcdf(const char & cdf, const double & prob).

The struct *f1* is

struct f1{double value; double der}.

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 25, 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 $0 \leq h \leq m-1$. 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 $0 \leq i < n_h$. 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.

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.

starttwoparamirt

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

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

As in invcdf, the cumulative distribution function is specified by cdf. The starting vector *starttwoparamirt* is based on the input matrix *responses* with m rows and n columns. Each row represents n binary items.

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. 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*.

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.

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 definition of *vecmat* is as in pack. 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

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

The struct *vecmat* is defined as in pack, 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.

normwt

The function normwt divides quadrature weights by the standard normal density to facilitate use with latent-structure models with latent variables that are normally distributed. The function declaration is

pw normwt(const pw & pwi).

The struct *normwt* has vector elements *normwt.points* and *normwt.weights*. The input *pwi* has elements *pwi.points* and *pwi.points*. The ordered quadrature points are in *normwt.points*. The corresponding weights are in *normwt.weights*. The result *normwt.points* is the same as *pwi.points*, but the weights *normwt.weights* are obtained from *pwi.weights* by dividing by the standard normal density at the corresponding points *pwi.points*.

qnormpw

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

pw qnormpw(const int & n).

The struct *qnormpw* has vector elements *qnormpw.points* and *qnormpw.weights*. The number of quadrature points is *n*. The ordered quadrature points are in *qnormpw.points*. The corresponding weights are in *qnormpw.weights*.

wmc

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

vecmat wmc(const vecmat & wx).

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.

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.

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.

Naylor, J. C., & Smith, A. F. M. (1982). Applications of a method for the efficient computation of posterior distributions. *Applied Statistics*, *31*, 214–225. https://doi.org/10.2307/2347995.

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.