Extracting and Unifying Semi-Elasticities and Effect Sizes from Studies with Binary and Categorical Dependent Variables

Geraldine Henningsen & Arne Henningsen

June 3, 2026

This is an addendum to our vignette "Extracting and Unifying Semi-Elasticities and Effect Sizes from Studies with Binary Dependent Variables."

1. Linear probability models and articles reporting marginal effects

1.1. Semi-elasticities from continuous explanatory variables (linear and quadratic)

Linear probability model with continuous explanatory variables:[1]

$$y = \beta_0 + \sum_{k=1}^{K} \beta_k x_k + u, \tag{1}$$

where $y \in \{0, 1\}$ is a binary dependent variable, $x = (x_1, \ldots, x_K)^\top$ is a vector of K continuous explanatory variables, $\beta = (\beta_0, \ldots, \beta_K)^\top$ is a vector of $K + 1$ unknown coefficients, and u is a random error term.

The semi-elasticity of the kth (continuous) explanatory variable is:

$$\epsilon_k = \frac{\partial E[y]}{\partial x_k} \, x_k = \beta_k \cdot x_k. \tag{2}$$

This semi-elasticity can be interpreted as: if the kth explanatory variable inceases by one percent, the probability that $y = 1$ increases by ϵ_k percentage points. This semi-elasticity depends on the value of the dependent variable x_k. One often calculates the semi-elasticity at the sample mean, i.e., $x_k = \bar{x}_k$.

If the model specification additionally includes a quadratic term of the explanatory variable k, e.g., $x_{k+1} = x_k^2$, the semi-elasticity of this explanatory variable is:

$$\epsilon_k = \frac{\partial E[y]}{\partial x_k} \, x_k = (\beta_k + 2 \, \beta_{k+1} \, x_k) \, x_k = \beta_k \, x_k + 2 \, \beta_{k+1} \, x_k^2. \tag{3}$$

[1] The following explanations are focused on linear probability models but apply equally to articles with probit or logit regressions that report the marginal effects.

An approximate standard error of the semi-elasticity defined in equation (3) can be obtained by using the Delta-method:

$$\operatorname{se}(\epsilon_k) = \sqrt{\frac{\partial \epsilon_k}{\partial(\beta_k, \beta_{k+1})} \operatorname{Var}(\beta_k, \beta_{k+1}) \frac{\partial \epsilon_k}{\partial(\beta_k, \beta_{k+1})^\top}} \tag{4}$$

$$= \sqrt{(x_k, 2\, x_k^2)\ \operatorname{Var}(\beta_k, \beta_{k+1})\ (x_k, 2\, x_k^2)^\top}, \tag{5}$$

where $\operatorname{se}(\epsilon_k)$ indicates the (approximate) standard error of the semi-elasticity ϵ_k and $\operatorname{Var}(\beta_k, \beta_{k+1})$ indicates the variance-covariance matrix of β_k and β_{k+1}.

As scientific publications usually do not report covariances between estimated coefficients, but only (at best) standard errors (or t-values, which can be used to calculate the standard errors), the covariance between the estimates of β_k and β_{k+1} is usually unknown. One can approximate equation (7) by assuming that the covariance between the estimates of β_k and β_{k+1}, i.e., the off-diagonal element(s) of $\operatorname{Var}(\beta_k, \beta_{k+1})$, is zero:

$$\operatorname{Var}(\beta_k, \beta_{k+1}) \approx \begin{bmatrix} \operatorname{se}(\beta_k)^2 & 0 \\ 0 & \operatorname{se}(\beta_{k+1})^2 \end{bmatrix}, \tag{6}$$

where $\operatorname{se}(\beta_k)$ and $\operatorname{se}(\beta_{k+1})$ are the standard errors of the estimates of β_k and β_{k+1}, respectively.

If there is no quadratic term of an explanatory variable x_k and, thus, the semi-elasticity is calculated according to equation (2), equation (5) simplifies to:

$$\operatorname{se}(\epsilon_k) = \operatorname{se}(\beta_k) \cdot x_k \tag{7}$$

The following function calculates the semi-elasticity ϵ_k according to equation (3) and its standard error according to equations (5) and (6):

```
linEla <- function( xCoef, xVal, xCoefSE = rep( NA, length( xCoef ) ) ){
  if( ! length( xCoef ) %in% c( 1, 2 ) ) {
    stop( "argument 'xCoef' must be a scalar or vector with 2 elements" )
  }
  if( length( xCoef ) != length( xCoefSE ) ) {
    stop( "arguments 'xCoef' and 'xCoefSE' must have the same length" )
  }
  if( length( xVal ) != 1 || !is.numeric( xVal ) ) {
    stop( "argument 'xVal' must be a single numeric value" )
  }
  if( length( xCoef ) == 1 ) {
    xCoef <- c( xCoef, 0 )
    xCoefSE <- c( xCoefSE, 0 )
  }
  semEla <- ( xCoef[1] + 2 * xCoef[2] * xVal ) * xVal
  derivCoef <- c( xVal, ifelse( xCoef[2] == 0, 0, 2 * xVal^2 ) )
  vcovCoef <- diag( xCoefSE^2 )
  semElaSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  result <- c( semEla = semEla, stdEr = semElaSE )
  return( result )
}
```

with argument `xCoef` $= (\beta_k, \beta_{k+1})^\top$, argument `xVal` $= x_k$, and an optional argument `xCoefSE` $= (\mathrm{se}(\beta_k), \mathrm{se}(\beta_{k+1}))^\top$. In case of model equations that are linear in x_k, the second elements of arguments `xCoef` and `xCoefSE` can be set to zero or can be omitted, i.e., `xCoef` $= \beta_k$ and `xCoefSE` $= \mathrm{se}(\beta_k)$, so that equation (3) simplifies to equation (2) and equation (5) simplifies to equation (7).

```
# Example
# model equation that is linear in x_k
ela1a <- linEla( 0.05, 23.4 )
ela1a

## semEla  stdEr
##   1.17     NA

ela1b <- linEla( 0.05, 23.4, 0.001 )
ela1b

## semEla  stdEr
## 1.1700 0.0234

all.equal( ela1b, linEla( c( 0.05, 0 ), 23.4, c( 0.001, 0 ) ) )

## [1] TRUE

# Example
# model equation that is quadratic in x_k
ela2a <- linEla( c( 0.05, -0.00002 ), 23.4 )
ela2a

##   semEla    stdEr
## 1.148098       NA

ela2b <- linEla( c( 0.05, -0.00002 ), 23.4, c( 0.001, 0.00002 ) )
ela2b

##     semEla      stdEr
## 1.14809760 0.03205113
```

1.2. Semi-elasticities from interval-coded explanatory variables

Linear probability model, where the kth explanatory variable is interval-coded:

$$y = \beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \sum_{m \in \{1,\ldots,M\} \setminus m^*} \delta_m D_m + u, \tag{8}$$

$$D_m = \begin{cases} 1 & \text{if } b_{m-1} < x_k \leq b_m \\ 0 & \text{otherwise} \end{cases} \quad \forall\, m = 1, \ldots, M, \tag{9}$$

where $y \in \{0,1\}$ is a binary dependent variable, $x = (x_1, \ldots, x_K)^\top$ is a vector of K continuous explanatory variables, whereas the actual values of one of these variables, x_k, are unobserved, $D = (D_1, \ldots, D_M)^\top$ is a vector of M dummy variables that indicates in which intervals the values of variable x_k fall, $b = (b_0, \ldots, b_M)^\top$ is a vector of the $M+1$ boundaries of the M intervals of variable x_k, $m^* \in \{1, \ldots, M\}$ is an arbitrary chosen interval that is used as 'base' interval in the regression, $\beta = (\beta_0, \ldots, \beta_K)^\top$ and $\delta = (\delta_1, \ldots, \delta_{m^*-1}, \delta_{m^*+1}, \ldots, \delta_M)^\top$ are vectors of $K+1$ and $M-1$, unknown coefficients, respectively, and u is a random error term. For convenience of further calculations, we define the (non-estimated) coefficient for the 'base' interval to be zero, i.e., $\delta_{m^*} = 0$.

The semi-elasticity of explanatory variable x_k:

$$\epsilon_k \equiv \frac{\partial E[y]}{\partial x_k}\, x_k \tag{10}$$

can be approximated 'around' each 'inner' interval boundary $b_1, \ldots, b_{M-1}$ by:

$$\epsilon_{km} \approx \frac{E[y|b_m < x_k \leq b_{m+1}] - E[y|b_{m-1} < x_k \leq b_m]}{E[x_k|b_m < x_k \leq b_{m+1}] - E[x_k|b_{m-1} < x_k \leq b_m]}\, b_m \tag{11}$$

$$= \frac{\delta_{m+1} - \delta_m}{E[x_k|b_m < x_k \leq b_{m+1}] - E[x_k|b_{m-1} < x_k \leq b_m]}\, b_m \tag{12}$$

$$\forall\, m = 1, \ldots, M-1.$$

It indicates the approximate increase in the probability of $y = 1$ (in percentage points) that is caused by an increase in the explanatory variable x_k by one percent around the interval border b_m.

Assuming:

$$E[x_k|b_{m-1} < x_k \leq b_m] \approx \frac{1}{2}\,(b_{m-1} + b_m)\,, \tag{13}$$

we get:

$$\epsilon_{km} \approx \frac{\delta_{m+1} - \delta_m}{\frac{1}{2}\,(b_m + b_{m+1}) - \frac{1}{2}\,(b_{m-1} + b_m)}\, b_m \tag{14}$$

$$= 2\, \frac{\delta_{m+1} - \delta_m}{b_{m+1} - b_{m-1}}\, b_m \tag{15}$$

$$\forall\, m = 1, \ldots, M-1$$

If b_M is infinity, we can assume:

$$E[x_k|b_{M-1} < x_k \leq b_M] \approx b_{M-1} + (b_{M-1} - b_{M-2}) \tag{16}$$

$$= 2\, b_{M-1} - b_{M-2} \tag{17}$$

$$= \frac{1}{2}\,(4\, b_{M-1} - 2\, b_{M-2}) \tag{18}$$

$$= \frac{1}{2}\,(b_{M-1} + 3\, b_{M-1} - 2\, b_{M-2})\,, \tag{19}$$

which is equivalent to setting:

$$b_M = 3\, b_{M-1} - 2\, b_{M-2}. \tag{20}$$

In order to aggregate the semi-elasticities $\epsilon_{k1}, \ldots, \epsilon_{k,M-1}$ that correspond to the 'inner' interval boundaries $b_1, \ldots, b_{M-1}$ to one overall semi-elasticity, we can calculate a weighted mean of the semi-elasticities $\epsilon_{k1}, \ldots, \epsilon_{k,M-1}$. We suggest to use the approximate proportions of observations that have a value of variable x_k that are 'around' each boundary $b_1, \ldots, b_{M-1}$ as weights:

$$w_m = \begin{cases} s_1 + \frac{1}{2} s_2 & \text{if } m = 1 \\ \frac{1}{2} \left(s_m + s_{m+1} \right) & \text{if } 2 \leq m \leq M-2 \\ \frac{1}{2} s_{M-1} + s_M & \text{if } m = M-1 \end{cases}, \tag{21}$$

where s_m is the proportion of observations that are in the mth interval, i.e., $b_{m-1} < x_k \leq b_m$. The weights sum up to one, i.e., $\sum_{m=1}^{M-1} w_m = 1$.

Thus, we can calculate the approximate average semi-elasticity by:

$$\epsilon_k \approx \sum_{m=1}^{M-1} w_m \, \epsilon_{km} \tag{22}$$

$$\begin{aligned} &= (2s_1 + s_2) \frac{\delta_2 - \delta_1}{b_2 - b_0} \, b_1 \\ &\quad + \sum_{m=2}^{M-2} (s_m + s_{m+1}) \frac{\delta_{m+1} - \delta_m}{b_{m+1} - b_{m-1}} \, b_m \\ &\quad + (s_{M-1} + 2s_M) \frac{\delta_M - \delta_{M-1}}{b_M - b_{M-2}} \, b_{M-1}. \end{aligned} \tag{23}$$

As argued before, if b_M is infinity, it can be set to the right-hand side of equation (20) or another value that seems to be appropriate for the analysed data set.

An approximate standard error of the semi-elasticity defined in equation (23) can be obtained by using the Delta-method:

$$\operatorname{se}(\epsilon_k) = \sqrt{\left(\frac{\partial \epsilon_k}{\partial \delta} \right)^\top \operatorname{Var}(\delta) \, \frac{\partial \epsilon_k}{\partial \delta}}, \tag{24}$$

where $\operatorname{se}(\epsilon_k)$ indicates the (approximate) standard error of ϵ_k and $\operatorname{Var}(\delta)$ indicates the variance-covariance matrix of the estimates of δ.

The nth element of the vector of partial derivatives $\partial\epsilon_k/\partial\delta$ can be obtained by:

$$\frac{\partial \epsilon_k}{\partial \delta_n} = \sum_{m=1}^{M-1} w_m \frac{\partial \epsilon_{km}}{\partial \delta_n} \tag{25}$$

$$= \begin{cases} w_1 \dfrac{\partial \epsilon_{k1}}{\partial \delta_1} & \text{if } n = 1 \\ w_{n-1} \dfrac{\partial \epsilon_{k,n-1}}{\partial \delta_n} + w_n \dfrac{\partial \epsilon_{kn}}{\partial \delta_n} & \text{if } 2 \leq n \leq M-1 \\ w_{M-1} \dfrac{\partial \epsilon_{k,M-1}}{\partial \delta_M} & \text{if } n = M \end{cases} \tag{26}$$

$$= \begin{cases} -2\, w_1 \dfrac{b_1}{b_2 - b_0} & \text{if } n = 1 \\ 2\, w_{n-1} \dfrac{b_{n-1}}{b_n - b_{n-2}} - 2\, w_n \dfrac{b_n}{b_{n+1} - b_{n-1}} & \text{if } 2 \leq n \leq M-1 \\ 2\, w_{M-1} \dfrac{b_{M-1}}{b_M - b_{M-2}} & \text{if } n = M \end{cases} \tag{27}$$

The following code defines a helper function that calculates the weights $w_1, \ldots, w_{M-1}$ according to equation (21):

```
elaIntWeights <- function( xShares ) {
  nInt <- length( xShares )
  weights <- rep( NA, nInt - 1 )
  for( m in 1:(nInt-1) ){
    weights[m] <- ifelse( m == 1, 1, 0.5 ) * xShares[m] +
      ifelse( m+1 == nInt, 1, 0.5 ) * xShares[m+1]
  }
  if( abs( sum( weights ) - 1 ) > 1e-5 ) {
    stop( "internal error: weights do not sum up to one" )
  }
  return( weights )
}
```

with argument `xShares` $= s = (s_1, \ldots, s_M)^\top$ the vector of proportion of observations in each interval.

The following code defines a helper function that checks the boundaries $b_0, \ldots, b_M$ and replaces b_M by the right-hand side of equation (20) if b_M is infinite:

```
elaIntBounds <- function( xBound, nInt, argName = "xBound" ) {
  if( length( xBound ) != nInt + 1 ) {
    stop( "argument '", argName, "' must be a vector with ", nInt + 1,
      " elements" )
  }
  if( any( xBound != sort( xBound ) ) ) {
    stop( "the elements of the vector specified by argument '", argName,
      "' must be in increasing order" )
```

```
  }
  if( max( table( xBound ) ) > 1 ) {
    stop( "the vector specified by argument '", argName,
      "' may not contain two (or more) elements with the same value" )
  }
  if( is.infinite( xBound[ nInt + 1 ] & nInt > 1 ) ) {
    xBound[ nInt + 1 ] <- 3 * xBound[ nInt ] - 2 * xBound[ nInt - 1 ]
  }
  return( xBound )
}
```

with argument `xBound` $= b = (b_0, \ldots, b_M)^\top$ the vector of boundaries, argument `nInt` $= M$ an integer that indicates the number of intervals, and optional argument `argName` a character string that can be used to modify the error message (if necessary).

Using the helper functions `elaIntWeights` (equation 21) and `elaIntBounds` (equation 20), the following code defines a function that calculates the semi-elasticity ϵ_k according to equations (15) and (22) and the respective standard error according to equations (24) and (27):

```
linElaInt <- function( xCoef, xShares, xBound,
                       xCoefSE = rep( NA, length( xCoef ) ) ){
  nInt <- length( xCoef )
  if( nInt < 2 || !is.vector( xCoef ) ) {
    stop( "argument 'xCoef' must be a vector with at least two elements" )
  }
  if( length( xCoefSE ) != nInt ) {
    stop( "arguments 'xCoef' and 'xCoefSE' must be vectors of the same length" )
  }
  if( length( xShares ) != nInt ) {
    stop( "arguments 'xCoef' and 'xShares' must be vectors of the same length" )
  }
  if( any( xShares < 0 ) ) {
    stop( "all shares in argument 'xShares' must be non-negative" )
  }
  if( abs( sum( xShares ) - 1 ) > 0.015 ) {
    stop( "the shares in argument 'xShares' must sum to one" )
  }
  # check 'xBound' and replace infinite values
  xBound <- elaIntBounds( xBound, nInt )
  # weights
  weights <- elaIntWeights( xShares )
  # semi-elasticities 'around' each inner boundary and their weights
  semElas <- rep( NA, nInt - 1 )
  for( m in 1:(nInt-1) ){
    semElas[m] <- 2 * ( xCoef[ m+1 ] - xCoef[ m ] ) * xBound[ m+1 ] /
      ( xBound[m+2] - xBound[m] )
```

```
  }
  # (average) semi-elasticity
  semElaAvg <- sum( semElas * weights )
  # derivatives of the (average) semi-elasticity wrt the coefficients
  derivCoef <- rep( NA, nInt )
  derivCoef[1] <-
   -2 * weights[1] * xBound[2] / ( xBound[3] - xBound[1] )
  derivCoef[nInt] <-
    2 * weights[nInt-1] * xBound[nInt] / ( xBound[nInt+1] - xBound[nInt-1] )
  if( nInt > 2 ) {
    for( n in 2:( nInt-1 ) ) {
      derivCoef[n] <-
        2 * weights[n-1] * xBound[n] / ( xBound[n+1] - xBound[n-1] ) -
        2 * weights[n]   * xBound[n+1] / ( xBound[n+2] - xBound[n] )
    }
  }
  # variance-covariance matrix of the coefficiencts
  vcovCoef <- diag( xCoefSE^2 )
  # standard error of the (average) semi-elasticity
  semElaSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  # prepare object that will be returned
  result <- c( semEla = semElaAvg, stdEr = semElaSE )
  return( result )
}
```

with argument **xCoef** $= \delta = (\delta_1, \ldots, \delta_M)^\top$, argument **xShares** $= s = (s_1, \ldots, s_M)^\top$, argument **xBound** $= b = (b_0, \ldots, b_M)^\top$, and an optional argument **xCoefSE** $= \delta = (\mathrm{se}(\delta_1), \ldots, \mathrm{se}(\delta_M))^\top$.

```
# Example
ela3a <- linElaInt( xCoef = c( 0, 0.22, 0.05, 0.6 ),
  xShares = c( 0.35, 0.4, 0.12, 0.13 ),
  xBound = c( 0, 500, 1000, 1500, Inf ) )
ela3a

## semEla  stdEr
## 0.0326     NA

# Example
ela3b <- linElaInt( xCoef = c( 0, 0.22, 0.05, 0.6 ),
  xShares = c( 0.35, 0.4, 0.12, 0.13 ),
  xBound = c( 0, 500, 1000, 1500, Inf ),
  xCoefSE = c( 0, 0.002, 0.005, 0.001 ) )
ela3b

##      semEla       stdEr
## 0.032600000 0.002600692
```

1.3. Effects of continuous variables when they change between discrete intervals

As in section 1.1, we assume a regression equation:

$$y = \beta_0 + \sum_{k=1}^{K} \beta_k x_k + u, \tag{28}$$

in which all explanatory variables $x_1, \ldots, x_K$ are continuous and also all other variables and coefficients are defined as above. In this section, we derive the (approximate) effects of a variable x_k on y if this variable changes between $M \geq 2$ discrete intervals (e.g., age categories), e.g., from a 'reference' interval l to an interval of interest m:

$$\begin{aligned}
E_{k,ml} =& E[y|b_{m-1} < x_k \leq b_m, x_{-k}] - E[y|b_{l-1} < x_k \leq b_l, x_{-k}] && (29)\\
=& \beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \beta_k E[x_k|b_{m-1} < x_k \leq b_m] && (30)\\
& - \beta_0 - \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j - \beta_k E[x_k|b_{l-1} < x_k \leq b_l]\\
=& \beta_k \left(\bar{x}_{km} - \bar{x}_{kl}\right), && (31)
\end{aligned}$$

where $x_{-k} = (x_1, \ldots, x_{k-1}, x_{k+1}, x_K)^\top$ is a vector of all explanatory variables except for x_k, $b_0, \ldots, b_M$ are the boundaries of the intervals of variable x_k, and

$$\bar{x}_{km} \equiv E[x_k|b_{m-1} < x_k \leq b_m] \; \forall \; m = 1, \ldots, M \tag{32}$$

are the expected values of variable x_k within specific intervals. If the expected values of variable x_k for specific intervals are unknown, it may be appropriate to approximate them by the mid-points of the respective interval boundaries (e.g., if the variable x_k has approximately a uniform distribution between the respective interval boundaries):

$$\bar{x}_{km} \approx \frac{b_{m-1} + b_m}{2} \; \forall \; m = 1, \ldots, M. \tag{33}$$

If the model specification additionally includes a quadratic term of the explanatory variable k, e.g., $x_{k+1} = x_k^2$, equations (29) to (31) change to:

$$\begin{aligned}
E_{k,ml} =& E[y|b_{m-1} < x_k \leq b_m, x_{-k}] - E[y|b_{l-1} < x_k \leq b_l, x_{-k}] && (34)\\
=& \beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus \{k,k+1\}} \beta_j x_j && (35)\\
& + \beta_k E[x_k|b_{m-1} < x_k \leq b_m] + \beta_{k+1} E[x_k^2|b_{m-1} < x_k \leq b_m]\\
& - \beta_0 - \sum_{j \in \{1,\ldots,K\} \setminus \{k,k+1\}} \beta_j x_j\\
& - \beta_k E[x_k|b_{l-1} < x_k \leq b_l] - \beta_{k+1} E[x_k^2|b_{l-1} < x_k \leq b_l]\\
=& \beta_k \left(\bar{x}_{km} - \bar{x}_{kl}\right) + \beta_{k+1} \left(\overline{x_{km}^2} - \overline{x_{kl}^2}\right) && (36)
\end{aligned}$$

with

$$\overline{x_{km}^2} \equiv E[x_k^2|b_{m-1} < x_k \leq b_m] \; \forall \; m = 1, \ldots, M. \tag{37}$$

If $E[x_k^2 | b_{m-1} < x_k \leq b_m]$ is unknown, it may be appropriate to approximate it by assuming that variable x_k has approximately a uniform distribution between each pair of subsequent interval boundaries so that its probability density function between boundaries b_{m-1} and b_m is $1/(b_m - b_{m-1})$:

$$\overline{x_{km}^2} \approx \int_{b_{m-1}}^{b_m} x_k^2 \, \frac{1}{b_m - b_{m-1}} \, d\, x_k \tag{38}$$

$$= \frac{1}{3} \, x_k^3 \, \frac{1}{b_m - b_{m-1}} \Bigg|_{b_{m-1}}^{b_m} \tag{39}$$

$$= \frac{1}{3} \, b_m^3 \, \frac{1}{b_m - b_{m-1}} - \frac{1}{3} \, b_{m-1}^3 \, \frac{1}{b_m - b_{m-1}} \tag{40}$$

$$= \frac{b_m^3 - b_{m-1}^3}{3 \, (b_m - b_{m-1})}. \tag{41}$$

An approximate standard error of the effect $E_{k,ml}$ calculated by equation (31) or (36) can be obtained—as above—with the Delta method:

$$\operatorname{se}(E_{k,ml}) = \sqrt{\left(\frac{\partial E_{k,ml}}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}} \right)^\top \operatorname{Var} \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix} \frac{\partial E_{k,ml}}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}}}, \tag{42}$$

where $\operatorname{se}(E_{k,ml})$ indicates the (approximate) standard error of $E_{k,ml}$, $\operatorname{Var}\left((\beta_k, \beta_{k+1})^\top\right)$ indicates the variance-covariance matrix of the estimates of β_k and β_{k+1}, the first element of the partial derivative of the effect $E_{k,ml}$ w.r.t. the coefficients is:

$$\frac{\partial E_{k,ml}}{\partial \beta_k} = \bar{x}_{km} - \bar{x}_{kl} \tag{43}$$

and the second element of this partial derivative is zero if there is no quadratic term of x_k (so that the effect is calculated by equation 31) and it is:

$$\frac{\partial E_{k,ml}}{\partial \beta_{k+1}} = \overline{x_{km}^2} - \overline{x_{kl}^2} \tag{44}$$

if there is a quadratic term of x_k (so that the effect is calculated by equation 36). If the covariance between β_k and β_{k+1} is unknown, one ccould assume that it is zero.

The following code defines a helper function that calculates $\overline{x_{km}^2}$ according to equation (41):

```
EXSquared <- function( lowerBound, upperBound ) {
  result <- ( upperBound^3 - lowerBound^3 )/( 3 * ( upperBound - lowerBound ) )
  return( result )
}
```

with arguments `lowerBound` $= b_{m-1}$ and `upperBound` $= b_m$ the lower and upper boundaries of the interval, respectively.

Using helper functions `EXSquared` (equation 41) and `elaIntBounds` (checking arguments `refBound` and `intBound`), the following function calculates the effect $E_{k,ml}$ and its approximate standard error $\mathrm{se}(E)$ according to equations (31), (33), (36), and (42) to (44):

```
linEffInt <- function( xCoef, refBound, intBound,
                       xCoefSE = rep( NA, length( xCoef ) ) ){
  if( ! length( xCoef ) %in% c( 1, 2 ) ) {
    stop( "argument 'xCoef' must be a scalar or vector with 2 elements" )
  }
  refBound <- elaIntBounds( refBound, 1, argName = "refBound" )
  intBound <- elaIntBounds( intBound, 1, argName = "intBound" )
  if( length( xCoef ) != length( xCoefSE ) ) {
    stop( "arguments 'xCoef' and 'xCoefSE' must have the same length" )
  }
  if( length( xCoef ) == 1 ) {
      xCoef <- c( xCoef, 0 )
      xCoefSE <- c( xCoefSE, 0 )
  }
  # difference between the xBars of the two intervals
  xDiff <- mean( intBound ) - mean( refBound )
  # difference between the xSquareBars of the two intervals
  xSquaredDiff <-
    EXSquared( intBound[1], intBound[2] ) -
    EXSquared( refBound[1], refBound[2] )
  # effect E_{k,ml}
  eff <-  xCoef[1] * xDiff + xCoef[2] * xSquaredDiff
  # partial derivative of E_{k,ml} w.r.t. the beta_k and beta_{k+1}
  derivCoef <- c( xDiff, ifelse( xCoef[2] == 0, 0, xSquaredDiff ) )
  # variance covariance of the coefficients (covariances set to zero)
  vcovCoef <- diag( xCoefSE^2 )
  # approximate standard error of the effect
  effSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  # object to be returned
  result <- c( effect = eff, stdEr = effSE )
  return( result )
}
```

with argument `xCoef` $= \beta_k$ or $(\beta_k, \beta_{k+1})^\top$ the coefficient(s) or marginal effect(s) of interest, `refBound` $= (b_{l-1}, b_l)$ the boundaries of the reference interval, `intBound` $= (b_{m-1}, b_m)$ the boundaries of the interval of interest, and optional argument `xCoefSE` $= \mathrm{se}(\beta_k)$ or $(\mathrm{se}(\beta_k), \mathrm{se}(\beta_{k+1}))^\top$ the standard error(s) of the coefficient(s) or marginal effect(s) of interest.

```
# Example
# model equation that is linear in x_k
eff1a <- linEffInt( 0.4, refBound = c( 19, 34 ), intBound = c( 35, 55 ) )
eff1a
```

```
## effect  stdEr
##    7.4     NA

eff1b <- linEffInt( 0.4, refBound = c( 19, 34 ), intBound = c( 35, 55 ),
                xCoefSE = 0.03 )
eff1b

## effect  stdEr
##  7.400  0.555

# Example
# model equation that is quadratic in x_k
eff2a <- linEffInt( c( 0.4, -0.0003 ),
                refBound = c( 19, 34 ), intBound = c( 35, 55 ) )
eff2a

## effect  stdEr
## 6.9988     NA

eff2b <- linEffInt( c( 0.4, -0.0003 ),
                refBound = c( 19, 34 ), intBound = c( 35, 55 ),
                xCoefSE = c( 0.002, 0.000001 ) )
eff2b

##     effect      stdEr
## 6.99880000 0.03702416
```

1.4. Grouping and re-basing categorical variables

We consider a regression model:

$$y = \beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \sum_{m \in \{1,\ldots,M\} \setminus m^*} \delta_m D_m + u, \tag{45}$$

$$D_m = \begin{cases} 1 & \text{if } x_k \in c_m \\ 0 & \text{otherwise} \end{cases} \quad \forall\, m = 1, \ldots, M, \tag{46}$$

where the variable of interest, x_k, is a categorical variable with M mutually exclusive categories $c_1, \ldots, c_M$ with $c_m \cap c_l = \emptyset \; \forall \; m \neq l$, category c_{m^*} is used as reference category, and all other variables and coefficients are defined as above. For notational simplification of the following derivations, we define the (non-estimated) coefficient of the reference category to be zero, i.e., $\delta_{m^*} \equiv 0$.

We want to obtain the effect of a change of variable x_k from a reference category c_r^* to a category of interest c_l^*:

$$E_{k,lr} = E[y|x_k \in c_l^*] - E[y|x_k \in c_r^*], \tag{47}$$

where categories c_r^* and/or c_l^* may comprise multiple original categories $c_1, \ldots, c_M$. Vectors $v_r = (v_{r1}, \ldots, v_{rM})^\top$ and $v_l = (v_{l1}, \ldots, v_{lM})^\top$ indicate, which of the original categories

$c_1, \ldots, c_M$ are included in categories c_r^* and c_l^*, respectively:

$$v_{nm} = \begin{cases} 1 & \text{if } c_m \in c_n^* \\ 0 & \text{if } c_m \notin c_n^* \end{cases} \forall\, m = 1, \ldots, M; n \in \{r, l\} \tag{48}$$

In the following, we derive the effect of a change of variable x_k from a reference category c_r^* to a category of interest c_l^*, $E_{k,lr}$ as defined in equation (47):

$$\begin{aligned} E_{k,lr} =& E[y|x_k \in c_l^*] - E[y|x_k \in c_r^*] && (49)\\ =& \beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m E[D_m|x_k \in c_l^*] && (50)\\ & - \beta_0 - \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j - \sum_{m=1}^{M} \delta_m E[D_m|x_k \in c_r^*] \\ =& \sum_{m=1}^{M} \delta_m \left(E[D_m|x_k \in c_l^*] - E[D_m|x_k \in c_r^*] \right) && (51)\\ =& \sum_{m=1}^{M} \delta_m \left(D_{ml} - D_{mr} \right) && (52) \end{aligned}$$

with

$$\begin{aligned} D_{mn} \equiv& E[D_m|x_k \in c_n^*] && (53)\\ =& \frac{P(D_m = 1 \cap x_k \in c_n^*)}{P(x_k \in c_n^*)} && (54)\\ =& \frac{E[D_m]\; v_{nm}}{P(x_k \in c_n^*)} && (55)\\ =& \frac{s_m v_{nm}}{\sum_{k=1}^{M} s_k v_{nk}} && (56)\\ & \forall\, m = 1, \ldots, M; n \in \{r, l\}, \end{aligned}$$

where $s_m = E[D_m]\ \forall\ m = 1, \ldots, M$ are the shares of observations, where variable x_k is in category c_m.

The delta method can be used to calculate approximate standard errors of $E_{k,lr}$. When setting all covariances between the coefficients to zero, an approximate standard error of $E_{k,lr}$ can be obtained by:

$$\text{se } (E_{k,lr}) \approx \sqrt{\sum_{m=1}^{M} \left(\text{se } (\delta_m)\ \left(D_{ml} - D_{mr} \right) \right)^2}, \tag{57}$$

where se (δ_m) are the standard errors of the estimates of δ_m. The standard error of the non-estimated coefficient of the reference category m^* is set to zero, i.e., se $(\delta_{m^*}) = 0$.

As an illustrative example, we assume that variable x_k has $M = 5$ categories, whereas the first category is used as reference category in the regression model, i.e., $m^* = 1$ and $\delta_1 \equiv 0$, and we want to measure the effect of variable x_k changing from the combined reference

category $\{3,4\}$ to the combined category of interest $\{1,2\}$ so that $v_r = (0,0,1,1,0)^\top$ and $v_l = (1,1,0,0,0)^\top$:

$$\begin{aligned} E_{k,\{1,2\}\,\{3,4\}} =& \delta_1 \left(\frac{s_1}{s_1+s_2} - \frac{0}{s_3+s_4} \right) + \delta_2 \left(\frac{s_2}{s_1+s_2} - \frac{0}{s_3+s_4} \right) \\ &+ \delta_3 \left(\frac{0}{s_1+s_2} - \frac{s_3}{s_3+s_4} \right) + \delta_4 \left(\frac{0}{s_1+s_2} - \frac{s_4}{s_3+s_4} \right) \\ &+ \delta_5 \left(\frac{0}{s_1+s_2} - \frac{0}{s_3+s_4} \right) \end{aligned} \tag{58}$$

$$= \frac{s_2 \cdot \delta_2}{s_1+s_2} - \frac{s_3 \cdot \delta_3}{s_3+s_4} - \frac{s_4 \cdot \delta_4}{s_3+s_4}. \tag{59}$$

with approximate standard error:

$$\text{se}\left(E_{k,\{1,2\}\,\{3,4\}}\right) = \sqrt{\left(\frac{s_2 \cdot \text{se}\left(\delta_2\right)}{s_1+s_2} \right)^2 - \left(\frac{s_3 \cdot \text{se}\left(\delta_3\right)}{s_3+s_4} \right)^2 - \left(\frac{s_4 \cdot \text{se}\left(\delta_4\right)}{s_3+s_4} \right)^2}. \tag{60}$$

The following function calculates the effect $E_{k,lr}$ and the its standard error according to equations (52), (56), and (57):

```
linEffGr <- function( xCoef, xShares, Group,
                      xCoefSE = rep( NA, length( xCoef ) ) ){
  if( sum( xShares ) > 1 ){
    stop( "the shares in argument 'xShares' sum up to a value larger than 1" )
  }
  if( length( xCoef ) != length( xShares ) ){
    stop( "arguments 'xCoef' and 'xShares' must have the same length" )
  }
  if( length( xCoef ) != length( Group ) ){
    stop( "arguments 'xCoef' and 'Group' must have the same length" )
  }
  if( ! all( Group %in% c( -1, 0, 1 ) ) ){
    stop( "all elements of argument 'Group' must be -1, 0, or 1" )
  }
  # D_mr
  DRef <- xShares * ( Group == -1 ) / sum( xShares[ Group == -1 ] )
  # D_ml
  DEffect <- xShares * ( Group == 1 ) / sum( xShares[ Group == 1 ] )
  # effect: sum of delta_m * ( D_ml - D_mr )
  effeG <- sum( xCoef * ( DEffect - DRef ) )
  # approximate standard error
  effeGSE <- sqrt( sum( ( xCoefSE * ( DEffect - DRef ) )^2 ) )
  result <- c( effect = effeG, stdEr = effeGSE )
  return( result )
}
```

with argument `xCoef` $= (\delta_1, \ldots, \delta_M)^\top$ a vector of coefficients of the categorical variable of interest, **where the coefficient of the reference group is set to 0**, argument `xShares`

$= (s_1, \ldots, s_M)^\top$ a vector of the corresponding shares of each category, argument `Group` $= (D_{1l} - D_{1r}, \ldots, D_{Ml} - D_{Mr})^\top$ a vector consisting of -1s, 0s, and 1s, where a -1 indicates that the category belongs to the reference category, a 1 indicates that the category belongs to the category of interest, and a zero indicates that the category is neither in the base category nor in the category of interest, and optional argument `xCoefSE` $= (\text{se}(\delta_1), \ldots, \text{se}(\delta_M))^\top$, **where the standard error of the coefficient of the reference group is set to 0**. Elements of arguments `xCoef`, `xShares`, `Group`, and `xCoefSE` that belong to a category that is neither in the reference category nor in the category of interest, i.e., categories m with $D_{mr} = D_{ml} = 0$, can be omitted but the ommission must be consistent for all four arguments.

```
# Example:
eff3a <- linEffGr( xCoef = c( 0, 0.2, 0.04, 0.06, 0.3 ),
                   xShares = c( 0.14, 0.35, 0.3, 0.01, 0.2 ),
                   Group = c( -1, 1, -1, -1, 0 ) )
eff3a

## effect  stdEr
##  0.172     NA

# Example:
eff3b <- linEffGr( xCoef = c( 0, 0.2, 0.04, 0.06, 0.3 ),
                   xShares = c( 0.14, 0.35, 0.3, 0.01, 0.2 ),
                   Group = c( -1, 1, -1, -1, 0 ),
                   xCoefSE = c( 0, 0.0001, 0.002, 0.05, 0.09 ))
eff3b

##      effect       stdEr
## 0.172000000 0.001738489

# Example:
eff3c <- linEffGr( xCoef = c( 0, 0.2, 0.04, 0.06 ),
                   xShares = c( 0.14, 0.35, 0.3, 0.01 ),
                   Group = c( -1, 1, -1, -1 ))
eff3c

## effect  stdEr
##  0.172     NA

# Example:
eff3d <- linEffGr( xCoef = c( 0, 0.2, 0.04, 0.06 ),
                   xShares = c( 0.14, 0.35, 0.3, 0.01 ),
                   Group = c( -1, 1, -1, -1 ),
                   xCoefSE = c( 0, 0.0001, 0.002, 0.05 ))
eff3d

##      effect       stdEr
## 0.172000000 0.001738489
```

2. Binary probit model, multivariate probit model, and ordered probit model

2.1. Semi-elasticities from continuous explanatory variables (linear and quadratic)

The (binary) probit model with continuous explanatory variables is specified as:

$$\Pr(y = 1|x) = \Phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right), \tag{61}$$

where $y \in \{0, 1\}$ is a binary dependent variable, $x = (x_1, \ldots, x_K)^\top$ is a vector of K continuous explanatory variables, $\beta = (\beta_0, \ldots, \beta_K)^\top$ is a vector of $K + 1$ unknown coefficients, and $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution.

The semi-elasticity of the kth (continuous) explanatory variable is:

$$\epsilon_k = \frac{\partial \Pr(y = 1)}{\partial x_k} \cdot x_k = \phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right) \cdot \beta_k x_k, \tag{62}$$

where $\phi(\cdot)$ is the probability density function of the standard normal distribution. The interpretation of the semi-elasticity ϵ_k is identical to the one described in section 1.1. The value of the semi-elasticity ϵ_k depends on the values of all explanatory variables $x_1, \ldots, x_K$. In order to obtain a 'representative' single value of this semi-elasticity, it is usually calculated at the sample mean values of the explanatory variables $\bar{x}_1, \ldots, \bar{x}_K$.

In case of binomial or multinomial probit models with two or more dependent variables $y_1, \ldots, y_N$, we only calculate the marginal effects of x_k on the unconditional expectations $E[y_n|x_1, \ldots, x_K]$ (and disregard marginal effects on the conditional expectations $E[y_n|x_1, \ldots, x_K, y_1, \ldots, y_{n-1}, y_{n+1}, \ldots, y_K)$. This has the advantage that we can ignore the interrelations between the regression models for the different dependent variables and obtain the semi-elasticities in the same way as for the univariate probit model, i.e., we can ignore the coefficients of the regression models for the other dependent variables as well as the coefficients of correlation between the error terms. Hence, equation (62) still applies.

The ordered probit model, where the dependent variable can have P distinct and strictly ordered values, i.e., $y \in \{1, \ldots, P\}$, can be specified as:

$$\Pr(y = p) = \Phi\left(\mu_p - \sum_{j=1}^{K} \beta_j x_j\right) - \Phi\left(\mu_{p-1} - \sum_{j=1}^{K} \beta_j x_j\right) \;\forall\, p \in \{1, \ldots, P\}, \tag{63}$$

where $\mu_0, \ldots, \mu_P$ are the break points, of which $\mu_0 = -\infty$, $\mu_P = \infty$, and $\mu_1, \ldots, \mu_{P-1}$ are unknown and, thus, need to be estimated. We create a new binary dependent variable y^* by dividing the P distinct values of the dependent variable y into two categories:

$$y^* = \begin{cases} 0 & \text{if } y \in \{1, \ldots, p^* - 1\} \\ 1 & \text{if } y \in \{p^*, \ldots, P\} \end{cases} \tag{64}$$

with $p^* \in \{2, \ldots, P\}$. This reduces the ordered probit model to a binary probit model:

$$\Pr(y^* = 1) = \Pr(y \in \{p^*, \ldots, P\}) \tag{65}$$

$$= \sum_{p=p^*}^{P} \Pr(y = p) \tag{66}$$

$$= \sum_{p=p^*}^{P} \left(\Phi\left(\mu_p - \sum_{j=1}^{K} \beta_j x_j \right) - \Phi\left(\mu_{p-1} - \sum_{j=1}^{K} \beta_j x_j \right) \right) \tag{67}$$

$$= \sum_{p=p^*}^{P} \Phi\left(\mu_p - \sum_{j=1}^{K} \beta_j x_j \right) - \sum_{p=p^*}^{P} \Phi\left(\mu_{P-1} - \sum_{j=1}^{K} \beta_j x_j \right) \tag{68}$$

$$= \sum_{p=p^*}^{P} \Phi\left(\mu_p - \sum_{j=1}^{K} \beta_j x_j \right) - \sum_{p=p^*-1}^{P-1} \Phi\left(\mu_p - \sum_{j=1}^{K} \beta_j x_j \right) \tag{69}$$

$$= \Phi\left(\mu_P - \sum_{j=1}^{K} \beta_j x_j \right) - \Phi\left(\mu_{p^*-1} - \sum_{j=1}^{K} \beta_j x_j \right) \tag{70}$$

$$= \Phi(\infty) - \Phi\left(\mu_{p^*-1} - \sum_{j=1}^{K} \beta_j x_j \right) \tag{71}$$

$$= 1 - \Phi\left(\mu_{p^*-1} - \sum_{j=1}^{K} \beta_j x_j \right) \tag{72}$$

$$= \Phi\left(-\mu_{p^*-1} + \sum_{j=1}^{K} \beta_j x_j \right), \tag{73}$$

where the intercept of the binary probit model is equal to the negative value of the break point that separates $y^* = 0$ from $y^* = 1$, i.e., $\beta_0 = -\mu_{p^*-1}$.[2] Hence, we can derive the semi-elasticity in the same way as for the binary probit model, i.e., by applying equation (62).

If the model specification additionally includes a quadratic term of the explanatory variable k, e.g., $x_{k+1} = x_k^2$, the semi-elasticity of this explanatory variable is:

$$\epsilon_k = \frac{\partial \Pr(y = 1)}{\partial x_k} \cdot x_k = \phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j \right) \cdot (\beta_k x_k + 2\beta_{k+1} x_k^2), \tag{74}$$

In order to calculate the standard errors of the marginal effects (and semi-elasticities) of a probit model, it is common to apply the Delta method as indicated in equation (5). But as in most cases the variance-covariance matrix of the regression coefficients is unknown, standard errors can only be approximated through a simplified Delta method that sets all covariances to zero. However, we conducted simulations that show that setting all covariances to zero leads to standard errors that are much larger than correctly calculated standard errors (i.e. using the

[2] If the ordered probit model is estimated with intercept, say, β_0^* and (for identification) by normalising the first (internal) break point to zero, i.e., $\mu_1 = 0$, the ordered probit model can be simplified to a binary probit model with intercept $\beta_0 = \beta_0^* - \mu_{p^*-1}$.

estimated covariances rather than setting them to zero). In contrast, our simulations indicated that simplifying the calculation of the standard errors by assuming that $\phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right)$ is a constant, i.e., assuming $\partial\phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right) / \partial\beta_n = 0 \; \forall \; n = 0, \ldots, K$, gives standard errors that are much closer to correctly calculated standard errors than including the full gradient vector of equation (62) with respect to the coefficients. Under this simplifying assumption, only one element of the gradient of equation (62) with respect to the coefficients is non-zero:

$$\frac{\partial \epsilon_k}{\partial \beta_k} = \phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right) \cdot x_k \tag{75}$$

and an approximate standard error of ϵ_k can be obtained by:

$$\begin{aligned} \operatorname{se}(\epsilon_k) &= \sqrt{\phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right) x_k \cdot \operatorname{Var}(\beta_k) \cdot \phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right) x_k} && (76) \\ &= \phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right) \cdot x_k \cdot \operatorname{se}(\beta_k), && (77) \end{aligned}$$

where $\operatorname{se}(\beta_k)$ is the standard error of the coefficient of interest.

If the variable of interest enters the regression equation in quadratic form as defined in equation (74), the simplified gradient vector has two non-zero elements:

$$\frac{\partial \epsilon_k}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}} = \begin{pmatrix} \phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right) \cdot x_k \\ \phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right) \cdot 2x_k^2 \end{pmatrix} \tag{78}$$

and an approximate standard error can be calculated by:

$$\operatorname{se}(\epsilon_k) = \sqrt{\left(\frac{\partial \epsilon_k}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}}\right)^{\top} \cdot \operatorname{Var}\begin{pmatrix} \beta_k & 0 \\ 0 & \beta_{k+1} \end{pmatrix} \cdot \left(\frac{\partial \epsilon_k}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}}\right)}. \tag{79}$$

Using the helper functions `checkXPos` (checking argument `xPos`, see appendix A) and `checkXBeta` (checking $\beta_0 + \sum_{j=1}^{K} \beta_j x_j$ for plausible values, see appendix A), the following function calculates the semi-elasticity, ϵ_k, and the corresponding standard error according to equations (62), (74), (78), and (79):

```
probEla <- function( allCoef, allXVal,
  allCoefSE = rep( NA, length( allCoef )), xPos ){
  nCoef <- length( allCoef )
  if( nCoef != length( allCoefSE ) ) {
    stop( "arguments 'allCoef' and 'allCoefSE' must have the same length" )
  }
```

```
  if( length( allCoef ) != length( allXVal ) ) {
    stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
  }
  checkXPos( xPos, minLength = 1, maxLength = 2, minVal = 1, maxVal = nCoef )
  if( length( xPos ) == 2 ){
    xCoef <- allCoef[ xPos ]
    xCoefSE <- allCoefSE[ xPos ]
    if( !isTRUE( all.equal( allXVal[ xPos[2] ], allXVal[ xPos[1] ]^2 ) ) ) {
      stop( "the value of 'allXVal[ xPos[2] ]' must be equal",
        "to the squared value of 'allXVal[ xPos[1] ]' " )
    }
  } else if( length( xPos ) == 1 ) {
    xCoef <- c( allCoef[ xPos ], 0 )
    xCoefSE <- c( allCoefSE[ xPos ], 0 )
  } else {
    stop( "argument 'xPos' must be a scalar or a vector with two elements" )
  }
  xVal <- allXVal[ xPos[ 1 ] ]
  xBeta <- sum( allCoef * allXVal )
  checkXBeta( xBeta )
  dfun <- pnorm( xBeta )
  semEla <- ( xCoef[ 1 ] + 2 * xCoef[ 2 ] * xVal ) * xVal * dfun
  derivCoef <- c( dfun * xVal,
    ifelse( length( xPos ) == 1, 0, dfun * 2 * xVal^2 ) )
  vcovCoef <- diag( xCoefSE^2 )
  semElaSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  result <- c( semEla = semEla, stdEr = semElaSE )
  return( result )
}
```

with argument `allCoef` $= (\beta_0, \ldots, \beta_K)^\top$, or alternatively in the case of an ordered probit model `allCoef` $= (-\mu_{p^*-1}, \beta_1, \ldots, \beta_K)^\top$, a vector of all coefficients; argument `allXVal` $= (1, x_1, \ldots, x_K)^\top$ a vector of the values of all corresponding explanatory variables including the intercept (e.g., their sample means); argument `xPos` $= k+1$ or $(k+1, k+2)^\top$ a scalar or vector with two elements that indicates the position of the variable of interest and its coefficient and eventually the position of the squared variable of interest and its coefficient in vectors `allXVal` and `allCoef`;[3] and optional argument `allCoefSE` $= (\text{se}(\beta_0), \ldots, \text{se}(\beta_K))^\top$ or alternatively in the case of an ordered probit model `allCoefSE` $= (\text{se}(\mu_{m^*-1}), \text{se}(\beta_1), \ldots, \text{se}(\beta_K))^\top$,[4] the standard errors of the coefficients.

```
# Example
ela4a <- probEla( c( 0.445, 0.03, 0.00002, 0.067, 0.89, 0.124 ),
```

[3] If argument `xPos` is a scalar, equation (74) simplifies to equation (62) and equations (78) and (79) simplify to equation (77).

[4] Note that $\text{se}(-\mu_{m^*-1}) = \sqrt{\text{Var}(-\mu_{m^*-1})} = \sqrt{(\partial(-\mu_{m^*-1})/\partial\mu_{m^*-1})^2 \text{ Var}(\mu_{m^*-1})} = \sqrt{(-1)^2 \text{Var}(\mu_{m^*-1})} = \sqrt{\text{Var}(\mu_{m^*-1})} = \text{se}(\mu_{m^*-1})$.

```
                c( 1, 2.34, 3.3, 3.3^2, 0.0, 0.987 ),
                xPos = 2 )
ela4a

##     semEla      stdEr
## 0.06417908         NA

# Example
ela4b <- probEla( c( 0.445, 0.03, 0.00002, 0.067, 0.89, 0.124 ),
                c( 1, 2.34, 3.3, 3.3^2, 0.0, 0.987 ),
                c( 0.032, 0.004, 0.00001, 0.034, 0.0009, 0.056 ),
                xPos = 2 )
ela4b

##     semEla      stdEr
## 0.06417908 0.00855721

# Example
ela4c <- probEla( c( 0.445, 0.03, 0.00002, 0.067, 0.89, 0.124 ),
                c( 1, 2.34, 3.3, 3.3^2, 0.0, 0.987 ),
                c( 0.032, 0.004, 0.00001, 0.034, 0.0009, 0.056 ),
                xPos = c( 3, 4 ))
ela4c

##   semEla    stdEr
## 1.334162 0.677007
```

2.2. Semi-elasticities from interval-coded explanatory variables

Probit model, where the kth explanatory variable is interval-coded:

$$\Pr(y=1|x) = \Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \sum_{m\in\{1,\ldots,M\}\setminus m^*} \delta_m D_m\right) \tag{80}$$

$$D_m = \begin{cases} 1 & \text{if } b_{m-1} < x_k \leq b_m \\ 0 & \text{otherwise} \end{cases} \quad \forall\, m = 1,\ldots,M, \tag{81}$$

where all variables and coefficients are as described in section 1.2.

As described in section 1.2, the semi-elasticities of the interval-coded variable x_k can be calculated as weighted averages by equation (22):

$$\epsilon_k \equiv \frac{\partial E[y]}{\partial x_k}\, x_k \approx \sum_{m=1}^{M-1} w_m\, \epsilon_{km} \tag{82}$$

with weights $w_1,\ldots,w_{M-1}$ as defined in equation (21). However, in contrast to section 1.2, the numerators of the semi-elasticities ϵ_{km} around each inner boundary $b_1,\ldots,b_{M-1}$ defined

in equation (11) cannot be simplified to differences between coefficients so that the derivations from equation (11) to equation (15) become:

$$\epsilon_{km} \approx \frac{E[y|b_m < x_k \leq b_{m+1}] - E[y|b_{m-1} < x_k \leq b_m]}{E[x_k|b_m < x_k \leq b_{m+1}] - E[x_k|b_{m-1} < x_k \leq b_m]}\, b_m \tag{83}$$

$$\approx 2\, \frac{\Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \delta_{m+1}\right) - \Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \delta_m\right)}{b_{m+1} - b_{m-1}} \cdot b_m \tag{84}$$

$$\forall\ m = 1, \ldots, M-1.$$

An approximate standard error of the semi-elasticity of interval-coded explanatory variables of probit models defined in equations (82) and (84) can be obtained by using the Delta-method:

$$\operatorname{se}(\epsilon_k) = \sqrt{\frac{\partial \epsilon_k}{\partial \left(\beta^\top\ \delta^\top\right)} \operatorname{Var}\begin{pmatrix}\beta\\ \delta\end{pmatrix} \frac{\partial \epsilon_k}{\partial \begin{pmatrix}\beta\\ \delta\end{pmatrix}}}, \tag{85}$$

where $\operatorname{se}(\epsilon_k)$ indicates the (approximate) standard error of ϵ_k, $\operatorname{Var}\left(\left(\beta^\top\ \delta^\top\right)^\top\right)$ indicates the variance-covariance matrix of the estimates of the coefficient vector $\left(\beta^\top\ \delta^\top\right)^\top$, and the gradient vector is:

$$\frac{\partial \epsilon_k}{\partial \begin{pmatrix}\beta\\ \delta\end{pmatrix}} = \sum_{m=1}^{M-1} w_m \frac{\partial \epsilon_{km}}{\partial \begin{pmatrix}\beta\\ \delta\end{pmatrix}} \tag{86}$$

with the elements of the gradient vectors $\partial\epsilon_{km}/\partial(\beta^\top\delta^\top)^\top$:

$$\frac{\partial \epsilon_{km}}{\partial \beta_0} = \frac{2\,(\phi_{m+1} - \phi_m)\, b_m}{b_{m+1} - b_{m-1}} \tag{87}$$

$$\frac{\partial \epsilon_{km}}{\partial \beta_j} = \frac{2\,(\phi_{m+1} - \phi_m)\, x_j\, b_m}{b_{m+1} - b_{m-1}}\ \forall\ j \in \{1, \ldots, K\} \setminus k \tag{88}$$

$$\frac{\partial \epsilon_{km}}{\partial \delta_m} = \frac{-2\,\phi_m\, b_m}{b_{m+1} - b_{m-1}} \tag{89}$$

$$\frac{\partial \epsilon_{km}}{\partial \delta_{m+1}} = \frac{2\,\phi_{m+1}\, b_m}{b_{m+1} - b_{m-1}} \tag{90}$$

$$\frac{\partial \epsilon_{km}}{\partial \delta_n} = 0\ \forall\ n \in \{1, \ldots, M\} \setminus \{m, m+1\} \tag{91}$$

with

$$\phi_m = \phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \delta_m\right)\ \forall\ m = 1, \ldots, M \tag{92}$$

In the case of the ordered probit, one can simply replace β_0 by $-\mu_{p^*-1}$ in equation (92) (see section 2.1):

$$\phi_m = \phi\left(-\mu_{p^*-1} + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \delta_m\right)\ \forall\ m = 1, \ldots, M. \tag{93}$$

Using the helper functions `elaIntWeights` (equation 21), `elaIntBounds` (equation 20), `checkXPos` (checking argument `xPos`, see appendix A), and `checkXBeta` (checking $\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \sum_{m \in \{1,\ldots,M\} \setminus m^*} \delta_m D_m$ for plausible values, see appendix A), and function `linElaInt` (section 1.2), the following code defines a function that calculates the semi-elasticity defined in equations (82) and (84) and its approximate standard error defined in equations (85) to (92):

```
probElaInt <- function( allCoef, allXVal, xPos, xBound,
                        allCoefSE = rep( NA, length( allCoef ) ) ){
  # number of coefficients
  nCoef <- length( allCoef )
  # number of intervals
  nInt <- length( xPos )
  # checking arguments
  if( length( allXVal ) != nCoef ) {
    stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
  }
  if( length( allCoefSE ) != nCoef ) {
    stop( "arguments 'allCoef' and 'allCoefSE' must have the same length" )
  }
  checkXPos( xPos, minLength = 2, maxLength = nCoef,
    minVal = 0, maxVal = nCoef, requiredVal = 0 )
  if( any( allXVal[ xPos ] < 0 ) ) {
    stop( "all elements of argument 'allXVal'",
      " that are indicated by argument 'xPos'",
      " (i.e., the shares of observations in each interval)",
      " must be non-negative" )
  }
  if( sum( allXVal[ xPos ] > 1 ) ) {
    stop( "the sum of the elements of argument 'allXVal'",
      " that are indicated by argument 'xPos'",
      " (i.e., the shares of observations in each interval)",
      " must not be larger than one" )
  }
  # check 'xBound' and replace infinite values
  xBound <- elaIntBounds( xBound, nInt )
  # vector of probabilities of y=1 for each interval and
  # vector of shares of observations in each interval
  xBeta <- shareVec <- rep( NA, nInt )
  for( i in 1:nInt ){
    allXValTemp <- replace( allXVal, xPos, 0 )
    if( xPos[i] != 0 ) {
      allXValTemp[ xPos[i] ] <- 1
      shareVec[ i ] <- allXVal[ xPos[ i ] ]
    }
    xBeta[ i ] <- sum( allCoef * allXValTemp )
  }
```

```
  shareVec[ xPos == 0 ] <- 1 - sum( shareVec[ xPos != 0 ] )
  checkXBeta( xBeta )
  phiVec <- pnorm( xBeta )
  # weights
  weights <- elaIntWeights( shareVec )
  # calculation of the semi-elasticity
  semEla <- linElaInt( phiVec, shareVec, xBound )
  ### calculation of its standard error
  # partial derivatives of each semi-elasticity around each boundary
  # w.r.t. all estimated coefficients
  gradM <- matrix( 0, nCoef, nInt - 1 )
  gradPhiVec <- dnorm( xBeta )
  for( m in 1:( nInt - 1 ) ) {
    gradM[ -xPos, m ] <- 2 * ( gradPhiVec[m+1] - gradPhiVec[m] ) *
        allXVal[ -xPos ] * xBound[m+1] / ( xBound[m+2] - xBound[m] )
    gradM[ xPos[m], m ] <- - 2 * gradPhiVec[m] * xBound[m+1] /
      ( xBound[m+2] - xBound[m] )
    gradM[ xPos[m+1], m ] <- 2 * gradPhiVec[m+1] * xBound[m+1] /
      ( xBound[m+2] - xBound[m] )
  }
  # partial derivative of the semi-elasticity
  # w.r.t. all estimated coefficients
  derivCoef <- rep( 0, nCoef )
  for( m in 1:( nInt - 1 ) ){
    derivCoef <- derivCoef + weights[m] * gradM[ , m ]
  }
  # variance-covariance matrix of the coefficiencts
  vcovCoef <- diag( allCoefSE^2 )
  # standard error of the (average) semi-elasticity
  semElaSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  # prepare object that will be returned
  result <- c( semEla[1], stdEr = semElaSE )
  return( result )
}
```

where argument `allCoef` $= (\beta_0, \ldots, \beta_{k-1},\ \delta_1, \ldots, \delta_{m^*-1},\ \delta_{m^*+1}, \ldots, \delta_M,\ \beta_{k+1}, \ldots, \beta_K)^\top$, or alternatively in the case of the ordered probit model `allCoef` $= (-\mu_{p^*-1}, \beta_1, \ldots, \beta_{k-1},\ \delta_1, \ldots, \delta_{m^*-1},\ \delta_{m^*+1}, \ldots, \delta_M,\ \beta_{k+1}, \ldots, \beta_K)^\top$, specifies the vector of all coefficients; argument `allXVal` $= (1, x_1, \ldots, x_{k-1},\ s_1, \ldots, s_{m^*-1}, s_{m^*+1}, \ldots, s_M,\ x_{k+1}, \ldots, x_K)^\top$ is a vector of corresponding values of the explanatory variables (including shares of observations in each interval of variable x_k except for the reference interval m^*); argument `xPos` $= (k+1, \ldots, k+m^*-1, 0, k+m^*, \ldots, k+M-1)$ is a vector indicating the positions of the coefficients $\delta_1, \ldots, \delta_M$ and shares $s_1, \ldots, s_M$ of each interval in the vectors `allCoef` and `allXVal`, whereas the position of the reference interval m^* is set to zero; argument `xBound` $= (b_0, \ldots, b_M)^\top$ indicates the boundaries of the intervals as in function `linElaInt`; and optional argument `allCoefSE` $= (\mathrm{se}(\beta_0), \ldots, \mathrm{se}(\beta_{k-1}), \mathrm{se}(\delta_1), \ldots, \mathrm{se}(\delta_{m^*-1}),$

$\text{se}(\delta_{m^*+1}), \ldots, \text{se}(\delta_M)$, $\text{se}(\beta_{k+1}), \ldots, \text{se}(\beta_K))^\top$, or alternatively in the case of an ordered probit model `allCoefSE` $= (\text{se}(\mu_{m^*-1}), \text{se}(\beta_1), \ldots, \text{se}(\beta_{k-1}), \text{se}(\delta_1), \ldots, \text{se}(\delta_{m^*-1}), \text{se}(\delta_{m^*+1}), \ldots, \text{se}(\delta_M), \text{se}(\beta_{k+1}), \ldots, \text{se}(\beta_K))^\top$, specifies the standard errors of all coefficients in `allCoef`.

```
# Example
ela5a <- probElaInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                     allXVal = c( 1, 0.4, 0.12, 0.13 ),
                     xPos = c( 2, 0, 3, 4 ),
                     xBound = c( 0, 500, 1000, 1500, Inf ))
ela5a

##      semEla       stdEr
## -0.03693385          NA

# Example
ela5b <- probElaInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                     allXVal = c( 1, 0.4, 0.12, 0.13 ),
                     xPos = c( 2, 0, 3, 4 ),
                     xBound = c( 0, 500, 1000, 1500, Inf ),
                     allCoefSE = c( 0.003, 0.045, 0.007, 0.009 ))
ela5b

##       semEla        stdEr
## -0.036933851  0.008957439
```

2.3. Effects of continuous variables when they change between discrete intervals

As in section 2.1 we assume the following model:

$$\Pr(y = 1|x) = \Phi\left(\beta_0 + \sum_{j=1}^{K} \beta_j x_j\right), \tag{94}$$

In order to derive the (approximate) effect of a continuous variable x_k on $Pr(y = 1)$ given

that x_k changes between $M \geq 2$ intervals, we modify equation (31) into:

$$E_{k,ml} = E[y|b_{m-1} < x_k \leq b_m] - E[y|b_{l-1} < x_k \leq b_l] \tag{95}$$

$$\approx \Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\backslash k} \beta_j x_j + \beta_k\, E[x_k|b_{m-1} < x_k \leq b_m]\right) \tag{96}$$

$$- \Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\backslash k} \beta_j x_j + \beta_k\, E[x_k|b_{l-1} < x_k \leq b_l]\right)$$

$$= \Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\backslash k} \beta_j x_j + \beta_k \bar{x}_{km}\right) \tag{97}$$

$$- \Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\backslash k} \beta_j x_j + \beta_k \bar{x}_{kl}\right),$$

where $\bar{x}_{km} \equiv E[x_k|b_{m-1} < x_k \leq b_m]\ \forall\ m = 1,\ldots,M$ can be approximated by the mid-points of the intervals as indicated by equation (33).[5]

For model specifications that additionally include a quadratic term of the explanatory variable, equation (36) modifies to:

$$E_{k,ml} = E[y|b_{m-1} < x_k \leq b_m, x_{-k}] - E[y|b_{l-1} < x_k \leq b_l, x_{-k}] \tag{98}$$

$$\approx \Phi\Bigg(\beta_0 + \sum_{j\in\{1,\ldots,K\}\backslash\{k,k+1\}} \beta_j x_j \tag{99}$$

$$+ \beta_k E[x_k|b_{m-1} < x_k \leq b_m] + \beta_{k+1} E[x_k^2|b_{m-1} < x_k \leq b_m]\Bigg)$$

$$- \Phi\Bigg(\beta_0 + \sum_{j\in\{1,\ldots,K\}\backslash\{k,k+1\}} \beta_j x_j$$

$$+ \beta_k E[x_k|b_{l-1} < x_k \leq b_l] + \beta_{k+1} E[x_k^2|b_{l-1} < x_k \leq b_l]\Bigg)$$

$$= \Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\backslash\{k,k+1\}} \beta_j x_j + \beta_k \bar{x}_{km} + \beta_{k+1}\overline{x^2}_{km}\right) \tag{100}$$

$$- \Phi\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\backslash\{k,k+1\}} \beta_j x_j + \beta_k \bar{x}_{kl} + \beta_{k+1}\overline{x^2}_{kl}\right),$$

where $\bar{x}_{km} \equiv E[x_k|b_{m-1} < x_k \leq b_m]$ and $\overline{x^2}_{km} \equiv E[x_k^2|b_{m-1} < x_k \leq b_m]\ \ \forall\ m = 1,\ldots,M$ remain the same as outlined in section 1.3.[6] If the values of $\bar{x}_{km}$ and $\overline{x^2}_{km}$ are unknown, they can again be approximated by equations (33) and (41), respectively.

[5] The derivation from equation (95) to equation (96) is approximate only. This derivation would be exact if the cumulative distribution function of the normal distribution $\Phi(\cdot)$ would be linear but this function is clearly non-linear.

[6] See footnote 5 regarding the approximate derivation from equation (98) to equation (99).

In order to calculate the approximate standard error of $E_{k,ml}$, we again apply the Delta method:

$$\text{se}\left(E_{k,ml}\right) = \sqrt{\left(\frac{\partial E_{k,ml}}{\partial \beta}\right)^\top \cdot \text{Var}\left(\beta\right) \cdot \frac{\partial E_{k,ml}}{\partial \beta}}, \tag{101}$$

where the elements of the gradient vector $\partial E_{k,ml} / \partial \beta$ are:

$$\frac{\partial E_{k,ml}}{\partial \beta_0} = \phi_m - \phi_l \tag{102}$$

$$\frac{\partial E_{k,ml}}{\partial \beta_j} = \left(\phi_m - \phi_l\right) x_j \; \forall \; j \in \{1, \ldots, k-1, k+2, \ldots, K\} \tag{103}$$

$$\frac{\partial E_{k,ml}}{\partial \beta_k} = \phi_m \; \bar{x}_{km} - \phi_l \; \bar{x}_{kl} \tag{104}$$

$$\frac{\partial E_{k,ml}}{\partial \beta_{k+1}} = \phi_m \; \overline{x^2}_{km} - \phi_l \; \overline{x^2}_{kl} \tag{105}$$

with $\phi_m \equiv \phi\left(\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus \{k,k+1\}} \beta_j x_j + \beta_k \bar{x}_{km} + \beta_{k+1} \overline{x^2}_{km}\right) \; \forall m = 1, \ldots, M$.

If the covariances between the estimated parameters are unknown, we can approximate equation (101) by assuming that the off-diagonal elements of the variance covariance matrix are zero.

Using helper functions `EXSquared` (equation 41), `elaIntBounds` (checking arguments `refBound` and `intBound`), `checkXPos` (checking argument `xPos`, see appendix A), and `checkXBeta` (checking $\beta_0 + \sum_{j=1}^{K} \beta_j x_j$ for plausible values, see appendix A), the following function calculates the effect and its standard error according to equations (97), (100), and (101):

```
probEffInt <- function( allCoef, allXVal, xPos, refBound, intBound,
                        allCoefSE = rep( NA, length( allCoef ) ) ){
  # number of coefficients
  nCoef <- length( allCoef )
  # check arguments
  if( length( allXVal ) != nCoef ){
    stop( "argument 'allCoef' and 'allXVal' must have the same length" )
  }
  if( length( allCoefSE ) != nCoef ){
    stop( "argument 'allCoef' and 'allCoefSE' must have the same length" )
  }
  checkXPos( xPos, minLength = 1, maxLength = 2, minVal = 1, maxVal = nCoef )
  refBound <- elaIntBounds( refBound, 1, argName = "refBound" )
  intBound <- elaIntBounds( intBound, 1, argName = "intBound" )
  if( any( !is.na( allXVal[ xPos ] ) ) ) {
    allXVal[ xPos ] <- NA
    warning( "values of argument 'allXVal[ xPos ]' are ignored",
      " (set these values to 'NA' to avoid this warning)" )
  }

  # calculate xBars
```

```
  intX <- mean( intBound )
  refX <- mean( refBound )
  if( length( xPos ) == 2 ) {
    intX <- c( intX, EXSquared( intBound[1], intBound[2] ) )
    refX <- c( refX, EXSquared( refBound[1], refBound[2] ) )
  }
  if( length( intX ) != length( xPos ) || length( refX ) != length( xPos ) ) {
    stop( "internal error: 'intX' or 'refX' does not have the expected length" )
  }
  # define X' * beta
  intXbeta <- sum( allCoef * replace( allXVal, xPos, intX ) )
  refXbeta <- sum( allCoef * replace( allXVal, xPos, refX ) )
  checkXBeta( c( intXbeta, refXbeta ) )
  # effect E_{k,ml}
  eff <- pnorm( intXbeta ) - pnorm( refXbeta )
  # partial derivative of E_{k,ml} w.r.t. all estimated coefficients
  derivCoef <- rep( NA, nCoef )
  derivCoef[ -xPos ] = ( dnorm( intXbeta ) - dnorm( refXbeta ) ) *
    allXVal[ -xPos ]
  derivCoef[ xPos ] = dnorm( intXbeta ) * intX - dnorm( refXbeta ) * refX
  # variance covariance of the coefficients (covariances set to zero)
  vcovCoef <- diag( allCoefSE^2 )
  # approximate standard error of the effect
  effSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  # object to be returned
  result <- c( effect = eff, stdEr = effSE )
  return( result )
}
```

where argument `allCoef` $= (\beta_0, \ldots, \beta_K)^\top$, or alternatively for an ordered probit model `allCoef` $= (-\mu_{p^*-1}, \beta_1, \ldots, \beta_K)^\top$, is a vector containing all coefficients; argument `allXVal` $= (1, x_1, \ldots, x_K)^\top$ is a vector containing all values of the corresponding explanatory variables; argument `xPos` $= k+1$ or $(k+1, k+2)^\top$ is a scalar or vector with two elements that indicates the position of the variable of interest and its coefficient and eventually the position of the squared variable of interest and its coefficient in vectors `allXVal` and `allCoef`; argument `refBound` $= (b_{l-1}, b_l)$ indicates the boundaries of the reference interval; argument `intBound` $= (b_{m-1}, b_m)$ indicates the boundaries of the interval of interest; and optional argument `allCoefSE` $= (\mathrm{se}(\beta_0), \ldots, \mathrm{se}(\beta_K))^\top$, or alternatively for ordered probit models `allCoefSE` $= (\mathrm{se}(\mu_{p^*-1}), \mathrm{se}(\beta_1), \ldots, \mathrm{se}(\beta_K))^\top$, is a vector of standard errors. Please note that the value of x_k and—in case of an additional quadratic term—also the value of $x_{k+1} = x_k^2$ in argument `allXVal`, i.e., `allXVal[xPos]`, are not used in the calculations of the effect $E_{k,ml}$ or in the calculation of its standard error and, thus, the value of x_k and—if present—also the value of $x_{k+1} = x_k^2$ should be set to `NA`.

```
# Example
eff4a <- probEffInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
```

```
                     allXVal = c( 1, NA, 0.16, 0.13 ),
                     xPos = 2,
                     refBound = c( 8, 12 ), intBound = c( 13, 15 ))
eff4a

##      effect       stdEr
## 0.004212184          NA

eff4b <- probEffInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                     allXVal = c( 1, NA, 0.16, 0.13 ),
                     xPos = 2,
                     refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                     allCoefSE = c( 0.003, 0.045, 0.007, 0.009 ) )
eff4b

##      effect       stdEr
## 0.004212184 0.005305154

# Example
eff5a <- probEffInt( allCoef = c( 0.33, 0.22, 0.05, 0.006 ),
                     allXVal = c( 1, NA, NA, 0.13 ),
                     xPos = c( 2, 3 ),
                     refBound = c( 8, 12 ), intBound = c( 13, 15 ))

## Warning in checkXBeta(c(intXbeta, refXbeta)): At least one x'beta has an
implausible value: 13.2274466666667, 7.59744666666667

eff5a

##       effect        stdEr
## 1.509903e-14           NA

eff5b <- probEffInt( allCoef = c( 0.33, 0.22, 0.05, 0.006 ),
                     allXVal = c( 1, NA, NA, 0.13 ),
                     xPos = c( 2, 3 ),
                     refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                     allCoefSE = c( 0.003, 0.045, 0.007, 0.009 ) )

## Warning in checkXBeta(c(intXbeta, refXbeta)): At least one x'beta has an
implausible value: 13.2274466666667, 7.59744666666667

eff5b

##       effect        stdEr
## 1.509903e-14 9.799652e-14
```

2.4. Grouping and re-basing categorical variables

We consider a regression model

$$\Pr(y=1|x) = \Phi\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j x_j + \sum_{m\in\{1,\dots,M\}\setminus m^*} \delta_m D_m\right) \tag{106}$$

$$D_m = \begin{cases} 1 & \text{if } x_k \in c_m \\ 0 & \text{otherwise} \end{cases} \quad \forall\, m = 1,\dots,M. \tag{107}$$

Like in section 1.4, the kth explanatory variable is coded into M mutually exclusive categories $c_1,\dots,c_M$, with $c_m \cap c_l = \emptyset\ \forall\ m \neq l$, and $D_1,\dots,D_M$ the corresponding dummy variables.

In the case of a probit regression, equation (52) modifies to:

$$E_{k,lr} = E[y|x_k \in c_l^*] - E[y|x_k \in c_r^*] \tag{108}$$

$$= \Phi\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m E[D_m|x_k \in c_l^*]\right) \tag{109}$$

$$- \Phi\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m E[D_m|x_k \in c_r^*]\right)$$

$$= \Phi\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{ml}\right) \tag{110}$$

$$- \Phi\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{mr}\right)$$

where D_{mn} is defined as in equation (56). In the case of an ordered probit regression, β_0 in equation (110) is again replaced by $-\mu_{p^*-1}$.

In order to calculate the approximate standard error of the new effect $E_{k,lr}$, we again apply the Delta method:

$$\operatorname{se}(E_{k,lr}) = \sqrt{\left(\frac{\partial E_{k,lr}}{\partial \begin{pmatrix}\beta\\ \delta\end{pmatrix}}\right)^{\top} \cdot \operatorname{Var}\begin{pmatrix}\beta\\ \delta\end{pmatrix} \cdot \frac{\partial E_{k,lr}}{\partial \begin{pmatrix}\beta\\ \delta\end{pmatrix}}}, \tag{111}$$

with the partial derivatives defined as:

$$\frac{\partial E_{k,lr}}{\partial \beta_0} = \phi_{ml}(\cdot) - \phi_{mr}(\cdot) \tag{112}$$

$$\frac{\partial E_{k,lr}}{\partial \beta_j} = (\phi_{ml}(\cdot) - \phi_{mr}(\cdot)) \cdot \bar{x}_j\ \forall\ j \in \{1,\dots,K\} \setminus k \tag{113}$$

$$\frac{\partial E_{k,lr}}{\partial \delta_m} = \phi_{ml}(\cdot) D_{ml} - \phi_{mr}(\cdot) D_{mr}\ \forall\ m = 1,\dots,M. \tag{114}$$

with

$$\phi_{ml}(\cdot) \equiv \phi \left(\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{ml} \right) \tag{115}$$

$$\phi_{mr}(\cdot) \equiv \phi \left(\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{mr} \right) \tag{116}$$

Using the same example as in section 1.4, equation (59) modifies to

$$E_{k,\{1,2\}\{3,4\}} = \Phi \left(\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \frac{\delta_1 \cdot s_1}{s_1 + s_2} + \frac{\delta_2 \cdot s_2}{s_1 + s_2} + \frac{\delta_3 \cdot 0}{s_1 + s_2} + \frac{\delta_4 \cdot 0}{s_1 + s_2} + \frac{\delta_5 \cdot 0}{s_1 + s_2} \right) \tag{117}$$

$$- \Phi \left(\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \frac{\delta_1 \cdot 0}{s_3 + s_4} + \frac{\delta_2 \cdot 0}{s_3 + s_4} + \frac{\delta_3 \cdot s_3}{s_3 + s_4} + \frac{\delta_4 \cdot s_4}{s_3 + s_4} + \frac{\delta_5 \cdot 0}{s_3 + s_4} \right)$$

$$= \Phi \left(\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \frac{\delta_1 \cdot s_1 + \delta_2 \cdot s_2}{s_1 + s_2} \right) \tag{118}$$

$$- \Phi \left(\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \frac{\delta_3 \cdot s_3 + \delta_4 \cdot s_4}{s_3 + s_4} \right)$$

The following function calculates the effect $E_{k,lr}$ and its standard error in accordance with equations (110) and (111) to (116):

```
probEffGr <- function( allCoef, allXVal, xPos, Group,
                       allCoefSE = rep( NA, length( allCoef ) ) ){
  nCoef <- length( allCoef )
  xShares <- allXVal[ xPos ]
  xCoef <- allCoef[ xPos ]
  if( sum( xShares ) > 1 ){
    stop( "the shares in argument 'xShares' sum up to a value larger than 1" )
  }
  if( length( xCoef ) != length( xShares ) ){
    stop( "arguments 'xCoef' and 'xShares' must have the same length" )
  }
  if( length( xCoef ) != length( Group ) ){
    stop( "arguments 'xCoef' and 'Group' must have the same length" )
  }
  if( ! all( Group %in% c( -1, 0, 1 ) ) ){
    stop( "all elements of argument 'Group' must be -1, 0, or 1" )
  }
  # D_mr
  DRef <- sum( xCoef[ Group == -1 ] * xShares[ Group == -1 ]) /
```

```
          sum( xShares[ Group == -1 ] )
  XBetaRef <- sum( allCoef[ -xPos ] * allXVal[ -xPos ]) + DRef
  # D_ml
  DEffect <- sum( xCoef[ Group == 1 ] * xShares[ Group == 1 ]) /
             sum( xShares[ Group == 1 ] )
  XBetaEffect <- sum( allCoef[ -xPos ] * allXVal[ -xPos ]) + DEffect
  # effect
  effeG <- pnorm( XBetaEffect ) - pnorm( XBetaRef )
  # partial derivative of E_{k,ml} w.r.t. all estimated coefficients
  derivCoef <- rep( NA, nCoef )
  derivCoef[ -xPos ] = ( dnorm( XBetaEffect ) - dnorm( XBetaRef ) ) *
                       allXVal[ -xPos ]
  derivCoef[ xPos ] = dnorm( XBetaEffect ) * DEffect - dnorm( XBetaRef ) * DRef
  # variance covariance of the coefficients (covariances set to zero)
  vcovCoef <- diag( allCoefSE^2 )
  # approximate standard error of the effect
  effeGSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  result <- c( effect = effeG, stdEr = effeGSE )
  return( result )
}
```

where argument `allCoef` $= (\beta_0, \ldots, \beta_{k-1}, \delta_1, \ldots, \delta_M, \beta_{k+1}, \ldots, \beta_K)^\top$ or alternatively `allCoef` $= (-\mu_{p^*-1}, \beta_1, \ldots, \beta_{k-1}, \delta_1, \ldots, \delta_M, \beta_{k+1}, \ldots, \beta_K)^\top$ are the coefficients of the probit regression or ordered probit regression, respectively, **with the coefficient of the reference group of the kth variable set to zero**, i.e., $\delta_{m^*} \equiv 0$; argument `allXVal` $= (x_1, \ldots, x_{k-1}, s_1, \ldots, s_M, x_{k+1}, \ldots, x_K)^\top$ are the mean values of the respective explanatory variables and the shares of each of the categories of the kth variable, **which includes the share of the reference group of the kth variable** (s_{m^*}); argument `xPos` is a vector of M elements indicating the positions of the coefficients $\delta_1, \ldots, \delta_M$ in argument `allCoef` and, equally, the positions of the shares $s_1, \ldots, s_M$ in argument `allXVal`; argument `Group` $= (D_{1l} - D_{1r}, \ldots, D_{Ml} - D_{Mr})^\top$ is a vector of M elements consisting of -1s, 0s, and 1s, where a -1 indicates that the category belongs to the new reference category, a 1 indicates that the category belongs to the new category of interest, and a zero indicates that the category belongs to neither; finally optional argument `allCoefSE` $= (\mathrm{se}(\beta_0), \ldots, \mathrm{se}(\beta_{k-1}), \mathrm{se}(\delta_1), \ldots, \mathrm{se}(\delta_M), \mathrm{se}(\beta_{k+1}), \ldots, \mathrm{se}(\beta_K))^\top$, or alternatively in the case of an ordered probit model `allCoefSE` $= (\mathrm{se}(\mu_{m^*-1}), \mathrm{se}(\beta_1), \ldots, \mathrm{se}(\beta_{k-1}), \mathrm{se}(\delta_1), \ldots, \mathrm{se}(\delta_M), \mathrm{se}(\beta_{k+1}), \ldots, \mathrm{se}(\beta_K))^\top$ are the standard errors of all coefficients of the probit regression, **where the standard error for the reference group is included as zero**, i.e., $\mathrm{se}(\delta_{m^*}) \equiv 0$. Elements of arguments `allCoef`, `allXVal`, `xPos`, `Group`, and `allCoefSE` that belong to a category that is neither in the reference category nor in the category of interest, i.e., categories m with $D_{mr} = D_{ml} = 0$, can be omitted but the ommission must be consistent for all five arguments.

```
# Example
Eff6a <- probEffGr( allCoef = c( 0.28, 0.003, 0.0075, 0, -0.034, -0.005, 0.89, -1.2 ),
                    allXVal = c( 1, 0.1, 0.3, 0.25, 0.15, 0.2, 2.34, 10.8 ),
                    xPos = c( 2:6 ), Group = c( 0, -1, -1, 1, 1 ) )
Eff6a
```

```
##        effect         stdEr
## -3.290613e-27            NA

# Example
Eff6b <- probEffGr( allCoef = c( 0.28, 0.003, 0.0075, 0, -0.034, -0.005, 0.89, -1.2 ),
                    allXVal = c( 1, 0.1, 0.3, 0.25, 0.15, 0.2, 2.34, 10.8 ),
                    xPos = c( 2:6 ), Group = c( 0, -1, -1, 1, 1 ),
                    allCoefSE = c( 0.03, 0.0001, 0.005, 0, 0.01, 0.004, 0.05, 0.8 ))
Eff6b

##        effect         stdEr
## -3.290613e-27  3.015010e-25

# the same examples but without categories that are neither
# in the new reference category nor in the new category of interest
all.equal( Eff6a,
  probEffGr( allCoef = c( 0.28, 0.0075, 0, -0.034, -0.005, 0.89, -1.2 ),
    allXVal = c( 1, 0.3, 0.25, 0.15, 0.2, 2.34, 10.8 ),
    xPos = c( 2:5 ), Group = c( -1, -1, 1, 1 ) ) )

## [1] TRUE

all.equal( Eff6b,
  probEffGr( allCoef = c( 0.28, 0.0075, 0, -0.034, -0.005, 0.89, -1.2 ),
    allXVal = c( 1, 0.3, 0.25, 0.15, 0.2, 2.34, 10.8 ),
    xPos = c( 2:5 ), Group = c( -1, -1, 1, 1 ),
    allCoefSE = c( 0.03, 0.005, 0, 0.01, 0.004, 0.05, 0.8 ) ) )

## [1] TRUE
```

3. Binary logit model, multinomial logit model, conditional logit model, and nested logit model

3.1. Semi-elasticities from continuous explanatory variables (linear and quadratic)

Binary logit model with continuous explanatory variables:

$$\Pr(y=1|x) = \frac{\exp\left(\beta_0 + \sum_{k=1}^{K} \beta_k x_k\right)}{1 + \exp\left(\beta_0 + \sum_{k=1}^{K} \beta_k x_k\right)}, \tag{119}$$

where $y \in \{0,1\}$ is a binary dependent variable, $x = (x_1, \ldots, x_K)^\top$ is a vector of K continuous explanatory variables, and $\beta = (\beta_0, \ldots, \beta_K)^\top$ is a vector of $K+1$ unknown coefficients.

The semi-elasticity of the kth (continuous) explanatory variable for the binary logit model is:

$$\epsilon_k = \frac{\partial \Pr(y=1)}{\partial \bar{x}_k} \cdot \bar{x}_k = \frac{\exp\left(\beta_0 + \sum_{k=1}^{K} \beta_k \bar{x}_k\right)}{\left(1 + \exp\left(\beta_0 + \sum_{k=1}^{K} \beta_k \bar{x}_k\right)\right)^2} \cdot \beta_k \bar{x}_k \tag{120}$$

The interpretation is equal to the one described in section 1.1.

If x_k also enters equation (119) in quadratic form the semi-elasticity modifies to

$$\epsilon_k = \frac{\exp\left(\beta_0 + \sum_{k\in\{1,\dots,K\}\setminus k+1} \beta_k \bar{x}_k + \beta_{k+1}\bar{x}_k^2\right)}{\left(1 + \exp\left(\beta_0 + \sum_{k\in\{1,\dots,K\}\setminus k+1} \beta_k \bar{x}_k + \beta_{k+1}\bar{x}_k^2\right)\right)^2} \cdot \left(\beta_k \bar{x}_k + 2\beta_{k+1}\bar{x}_k^2\right) \tag{121}$$

To calculate the (approximate) standard error of ϵ_k, we again use the simplified Delta method as described in section 2.1, i.e., we assume the covariances to be zero and $\frac{\exp\left(\beta_0+\sum_{k=1}^K \beta_k \bar{x}_k\right)}{\left(1+\exp\left(\beta_0+\sum_{k=1}^K \beta_k \bar{x}_k\right)\right)^2}$ to be a constant. The standard error then calculates as

$$\operatorname{se}(\epsilon_k) = \sqrt{\frac{\exp\left(\beta_0 + \sum_{k=1}^K \beta_k \bar{x}_k\right)}{\left(1 + \exp\left(\beta_0 + \sum_{k=1}^K \beta_k \bar{x}_k\right)\right)^2} \cdot \bar{x}_k \cdot \operatorname{Var}(\beta_k) \cdot \frac{\exp\left(\beta_0 + \sum_{k=1}^K \beta_k \bar{x}_k\right)}{\left(1 + \exp\left(\beta_0 + \sum_{k=1}^K \beta_k \bar{x}_k\right)\right)^2} \cdot \bar{x}_k} \tag{122}$$

$$= \frac{\exp\left(\beta_0 + \sum_{k=1}^K \beta_k \bar{x}_k\right)}{\left(1 + \exp\left(\beta_0 + \sum_{k=1}^K \beta_k \bar{x}_k\right)\right)^2} \cdot \bar{x}_k \cdot \operatorname{se}(\beta_k) \tag{123}$$

and in case of a quadratic covariate $\bar{x}_k^2$

$$\frac{\partial \epsilon_k}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}} = \begin{pmatrix} \dfrac{\exp\left(\beta_0 + \sum_{k\in\{1,\dots,K\}\setminus k+1} \beta_k \bar{x}_k + \beta_{k+1}\bar{x}_k^2\right)}{\left(1 + \exp\left(\beta_0 + \sum_{k\in\{1,\dots,K\}\setminus k+1} \beta_k \bar{x}_k + \beta_{k+1}\bar{x}_k^2\right)\right)^2} \cdot \bar{x}_k \\ \dfrac{\exp\left(\beta_0 + \sum_{k\in\{1,\dots,K\}\setminus k+1} \beta_k \bar{x}_k + \beta_{k+1}\bar{x}_k^2\right)}{\left(1 + \exp\left(\beta_0 + \sum_{k\in\{1,\dots,K\}\setminus k+1} \beta_k \bar{x}_k + \beta_{k+1}\bar{x}_k^2\right)\right)^2} \cdot 2\bar{x}_k^2 \end{pmatrix} \tag{124}$$

$$\operatorname{se}(\epsilon_k) = \sqrt{\left(\frac{\partial \epsilon_k}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}}\right)^\top \cdot \operatorname{Var}\begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix} \cdot \frac{\partial \epsilon_k}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}}} \tag{125}$$

The multinomial logit model with continuous explanatory variables:

$$\Pr(y = p|x) = \pi_p = \frac{\exp(\beta_{0,p} + \sum_{k=1}^K \beta_{k,p} x_k))}{1 + \sum_{p\in\{1,\dots,P\}\setminus p^*} \exp(\beta_{0,p} + \sum_{k=1}^K \beta_{k,p} x_k))} \tag{126}$$

where $y \in \{1, \dots, P\}$ is a categorical dependent variable of which the base category, $p*$, is set to zero, $x = (x_1, \dots, x_K)^\top$ is a vector of K continuous explanatory variables describing the characteristics of the choice taker and $\beta = (\beta_{0,p}, \dots, \beta_{K,p})^\top$ is a vector of $K+1$ unknown coefficients.

The semi-elasticity of the kth (continuous) explanatory variable for the multinomial logit model is:

$$\begin{aligned} \epsilon_{k,p} &= \frac{\partial \Pr(y = p)}{\partial \bar{x}_k} \cdot \bar{x}_k \\ &= \pi_p \left(\frac{\beta_{k,p}}{[1 + \sum_{p\in\{1,\dots,P\}\setminus p^*} \exp(\beta_{0,p} + \sum_{k=1}^K \beta_{k,p} x_k)]} + \sum_{o\in\{1,\dots,P\}\setminus\{p,p^*\}} (\beta_{k,p} - \beta_{k,o})\pi_o \right) \cdot \bar{x}_k \end{aligned} \tag{127}$$

This semi-elasticity can be interpreted as: if the k,pth explanatory variable inceases by one percent, the probability that y belongs to category p instead of the base category, increases by $\epsilon_{k,p}$ percentage points.

If x_k also enters equation (134) in quadratic form,

$$\Pr(y=p|x)=\pi_p=\frac{\exp(\beta_{0,p}+\sum_{k\in\{1,\dots,K\}\backslash k+1}\beta_{k,p}\bar{x}_k+\beta_{k+1,p}\bar{x}_k^2))}{1+\sum_{p\in\{1,\dots,P\}\backslash p^*}\exp(\beta_{0,p}+\sum_{k\in\{1,\dots,K\}\backslash k+1}\beta_{k,p}\bar{x}_k+\beta_{k+1,p}\bar{x}_k^2))} \tag{128}$$

equation (127) modifies to

$$\begin{aligned}\epsilon_{k,p}=&\frac{\partial\Pr(y=p)}{\partial\bar{x}_k}\cdot\bar{x}_k\\ =&\pi_p\left(\frac{(\beta_{k,p}+2\beta_{k+1,p}\bar{x}_k)}{[1+\sum_{p\in\{1,\dots,P\}\backslash p^*}\exp(\beta_{0,p}+\sum_{k\in\{1,\dots,K\}\backslash k+1}\beta_{k,p}\bar{x}_k+\beta_{k+1,p}\bar{x}_k^2)]}\right.\\ &\left.+\sum_{o\in\{1,\dots,P\}\backslash\{p,p^*\}}((\beta_{k,p}+2\beta_{k+1,p}\bar{x}_k)-(\beta_{k,o}+2\beta_{k+1,0}\bar{x}_k))\pi_o\right)\cdot\bar{x}_k\end{aligned} \tag{129}$$

To calculate the (approximate) standard error $\epsilon_{k,p}$, we use a simplified Delta method, where we again set all covariances between the βs to zero and assume for the most part fixed factors for the remaining terms. Hence,

$$\begin{aligned}\frac{\partial\epsilon_{k,p}}{\partial\beta_{k,p}}=&\frac{\exp(\beta_{0,p}+\sum_{k=1}^K\beta_{k,p}x_k))}{[1+\sum_{p\in\{1,\dots,P\}\backslash p^*}\exp(\beta_{0,p}+\sum_{k=1}^K\beta_{k,p}x_k)]^2}\bar{x}_k\\ &+\pi_p\sum_{o\in\{1,\dots,P\}\backslash\{p,p^*\}}(\beta_{k,p}-\beta_{k,o})\pi_o\cdot\bar{x}_k\end{aligned} \tag{130}$$

$$\begin{aligned}\operatorname{se}(\epsilon_{k,p})&=\sqrt{\left(\frac{\partial\epsilon_{k,p}}{\partial\beta_{k,p}}\right)^\top\operatorname{Var}(\beta_{k,p})\frac{\partial\epsilon_{k,p}}{\partial\beta_{k,p}}}\\ &=\sqrt{\frac{\partial\epsilon_{k,p}}{\partial\beta_{k,p}}}\operatorname{se}(\beta_{k,p})\end{aligned} \tag{131}$$

and in case of a quadratic covariate $\bar{x}_k^2$

$$\frac{\partial \epsilon_{k,p}}{\partial \begin{pmatrix} \beta_{k,p} \\ \beta_{k+1,p} \end{pmatrix}} = \begin{pmatrix} \frac{\exp(\beta_{0,p} + \sum_{k \in \{1,\dots,K\} \setminus k+1} \beta_{k,p} \bar{x}_k + \beta_{k+1,p} \bar{x}_k^2)))}{[1 + \sum_{p \in \{1,\dots,P\} \setminus p^*} \exp(\beta_{0,p} + \sum_{k \in \{1,\dots,K\} \setminus k+1} \beta_{k,p} \bar{x}_k + \beta_{k+1,p} \bar{x}_k^2)]^2} \bar{x}_k \\ + \pi_p \sum_{o \in \{1,\dots,P\} \setminus \{p,p^*\}} ((\beta_{k,p} + 2\beta_{k+1,p} \bar{x}_k) - (\beta_{k,o} + 2\beta_{k+1,0} \bar{x}_k)) \pi_o \cdot \bar{x}_k \\ \\ \frac{\exp(\beta_{0,p} + \sum_{k \in \{1,\dots,K\} \setminus k+1} \beta_{k,p} \bar{x}_k + \beta_{k+1,p} \bar{x}_k^2)))}{[1 + \sum_{p \in \{1,\dots,P\} \setminus p^*} \exp(\beta_{0,p} + \sum_{k \in \{1,\dots,K\} \setminus k+1} \beta_{k,p} \bar{x}_k + \beta_{k+1,p} \bar{x}_k^2)]^2} 2\bar{x}_k^2 \\ + \pi_p \sum_{o \in \{1,\dots,P\} \setminus \{p,p^*\}} ((\beta_{k,p} + 2\beta_{k+1,p} \bar{x}_k) - (\beta_{k,o} + 2\beta_{k+1,0} \bar{x}_k)) \pi_o \cdot 2\bar{x}_k^2 \end{pmatrix} \tag{132}$$

$$\operatorname{se}(\epsilon_{k,p}) = \sqrt{\left(\frac{\partial \epsilon_{k,p}}{\partial \begin{pmatrix} \beta_{k,p} \\ \beta_{k+1,p} \end{pmatrix}} \right)^\top \cdot \operatorname{Var} \begin{pmatrix} \beta_{k,p} \\ \beta_{k+1,p} \end{pmatrix} \cdot \frac{\partial \epsilon_{k,p}}{\partial \begin{pmatrix} \beta_{k,p} \\ \beta_{k+1,p} \end{pmatrix}}} \tag{133}$$

Conditional logit model with continuous explanatory variables:

$$\Pr(y = p|z) = \pi_p = \frac{\exp(\gamma_0 + \sum_{t=1}^T \gamma_t z_{t,p}))}{\sum_{p \in C} \exp(\gamma_0 + \sum_{t=1}^T \gamma_t z_{t,p}))} \tag{134}$$

where $y \in \{1, \dots, P\}$ is a categorical dependent variable, $z = (z_{1,p}, \dots, z_{T,p})^\top$ is a vector of T continuous explanatory variables describing the characteristics of each of the P alternatives in the choice set C, and $\gamma = (\gamma_0, \dots, \gamma_T)^\top$ is a vector of $T + 1$ unknown coefficients.

The semi-elasticity of the tth (continuous) explanatory variable for the conditional logit model is:

$$\begin{aligned} \epsilon_{t,p} &= \frac{\partial \Pr(y = p)}{\partial \bar{z}_{t,p}} \cdot \bar{z}_{t,p} \\ &= (\pi_p - \pi_p^2) \cdot \gamma_t \bar{z}_{t,p} \end{aligned} \tag{135}$$

This semi-elasticity can be interpreted as: if the tth explanatory variable inceases by one percent, the probability that y belongs to category j increases by $\epsilon_{t,p}$ percentage points.

If $z_{t,p}$ is also included in quadratic form, equation (135) modifies to

$$\begin{aligned} \epsilon_{t,p} &= \frac{\partial \Pr(y = p)}{\partial \bar{z}_{t,p}} \cdot \bar{z}_{t,p} \\ &= (\pi_p - \pi_p^2) \cdot (\gamma_t \bar{z}_{t,p} + 2\gamma_{t+1} \bar{z}_{t,p}^2) \end{aligned} \tag{136}$$

In order to calculate approximate standard errors in the case of the conditional logit regression, equation (123) modifies to

$$\operatorname{se}(\epsilon_{t,p}) = (\pi_p - \pi_p^2) z_{t,p} \cdot \operatorname{se}(\gamma_t) \tag{137}$$

and in case of a quadratic covariate $\bar{z}_t^2$

$$\frac{\partial \epsilon_{t,p}}{\partial \begin{pmatrix} \gamma_t \\ \gamma_{t+1} \end{pmatrix}} = \begin{pmatrix} (\pi_p - \pi_p^2) z_{t,p} \\ \\ (\pi_p - \pi_p^2) 2 z_{t,p}^2 \end{pmatrix} \tag{138}$$

$$\text{se}\left(\epsilon_{t,p}\right) = \sqrt{\left(\frac{\partial \epsilon_{t,p}}{\partial \begin{pmatrix} \gamma_t \\ \gamma_{t+1} \end{pmatrix}}\right)^\top \cdot \text{Var}\begin{pmatrix} \gamma_t \\ \gamma_{t+1} \end{pmatrix} \cdot \frac{\partial \epsilon_{t,p}}{\partial \begin{pmatrix} \gamma_t \\ \gamma_{t+1} \end{pmatrix}}} \tag{139}$$

For the nested logit model with continuous explanatory variables at both the branch and the twig level,[7] we have to differentiate between the probability of choice of the oth branch, B_o,

$$\Pr(y \in B_o | x) = \pi_o = \frac{\exp(\gamma_0 + \sum_{t=1}^T \gamma_t z_{t,o} + \lambda_o IV_o)}{\sum_{o=1}^O \exp(\gamma_0 + \sum_{t=1}^T \gamma_t z_{t,o} + \lambda_o IV_o)} \tag{140}$$

$$\text{with} \quad IV_o = \ln \sum_{p \in B_o} \exp\left(\beta_0/\lambda_o + \sum_{k=1}^K \beta_k/\lambda_o x_{k,p}\right) \tag{141}$$

and the combined probablity of choosing the pth alternative in the oth branch

$$\begin{aligned} \Pr(y = p, y \in B_o) = & \Pr(y = p | x, y \in B_o) \cdot \Pr(y \in B_o | x) = \pi_p \cdot \pi_o \\ & = \frac{\exp(\beta_0/\lambda_o + \sum_{k=1}^K \beta_k/\lambda_o x_{k,p})}{\sum_{p \in B_o} \exp(\beta_0/\lambda_o + \sum_{k=1}^K \beta_k/\lambda_o x_{k,p})} \cdot \frac{\exp(\gamma_0 + \sum_{t=1}^T \gamma_t z_{t,o} + \lambda_o IV_o)}{\sum_{o=1}^O \exp(\gamma_0 + \sum_{t=1}^T \gamma_t z_{t,o} + \lambda_o IV_o)} \end{aligned} \tag{142}$$

where $z_o = (1, z_{1,o}, \ldots, z_{T,o})^\top$ are the branch specific sets of characterics (which can be empty sets), $\gamma = (\gamma_0, \ldots, \gamma_T)^\top$ are the coefficients of the branch specific characteristics, $x_p = (1, x_{1,p}, \ldots, x_{K,p})^\top$ are variables describing the characteristics of the P_o alternatives of B_o and/or of the choice maker, and $\beta = (\beta_0, \ldots, \beta_K)^\top$ are the corresponding coefficients, and $\lambda = (\lambda_1, \ldots, \lambda_O)^\top$ are the parameters of the correlation coefficients $(1 - \lambda_o)$ between the P_o branch specific alternatives.

At the branch level, the semi-elasticity for a variable $z_{t,o}$ can be derived as:

$$\epsilon_{t,o} = (\pi_o - \pi_o^2) \cdot \gamma_t \bar{z}_{t,o} \tag{143}$$

and if $z_{t,o}$ enters the equation in quadratic form:

$$\epsilon_{t,o} = (\pi_o - \pi_o^2) \cdot (\gamma_t \bar{z}_{t,o} + 2\gamma_{t+1} \bar{z}_{t,o}^2) \tag{144}$$

[7] We restrict our analysis on a nesting structure with two levels, branches and twigs, as these seem most common in the applied empirical work. For nested logits with higher levelled nesting structures, the formulas must be modified accordingly. However, the reader should not that an analytical solution for nested logits with higher levelled nesting structures might become very complicated and computational solutions might be preferred.

whilst the semi-elasticity for $x_{k,p}$ wrt the combined probability is:

$$\epsilon_{k,p} = \left(\pi_o(\pi_p - \pi_p^2)\frac{1}{\lambda_o} + \pi_p^2(\pi_o - \pi_o^2)\lambda_o IV_o \right) \cdot \beta_k \bar{x}_{k,p} \tag{145}$$

and for the quadratic form

$$\epsilon_{k,p} = \left(\pi_o(\pi_p - \pi_p^2)\frac{1}{\lambda_o} + \pi_p^2(\pi_o - \pi_o^2)\lambda_o IV_o \right) \cdot (\beta_k \bar{x}_{k,p} + 2\beta_{k+1}\bar{x}_{k,p}^2) \tag{146}$$

In order to calculate approximate standard errors for the semi-elasticity at the branch level (equation (140)), equation (123) modifies to

$$\operatorname{se}(\epsilon_{t,o}) = (\pi_o - \pi_o^2) \cdot z_{t,o} \cdot \operatorname{se}(\gamma_t) \tag{147}$$

and in case of a quadratic covariate $\bar{z}_{t,o}^2$

$$\frac{\partial \epsilon_{t,o}}{\partial \begin{pmatrix} \gamma_t \\ \gamma_{t+1} \end{pmatrix}} = \begin{pmatrix} (\pi_o - \pi_o^2) z_{t,o} \\ (\pi_o - \pi_o^2) 2 z_{t,o}^2 \end{pmatrix} \tag{148}$$

$$\operatorname{se}(\epsilon_{t,o}) = \sqrt{\left(\frac{\partial \epsilon_{t,o}}{\partial \begin{pmatrix} \gamma_t \\ \gamma_{t+1} \end{pmatrix}} \right)^\top \cdot \operatorname{Var} \begin{pmatrix} \gamma_t \\ \gamma_{t+1} \end{pmatrix} \cdot \frac{\partial \epsilon_{t,o}}{\partial \begin{pmatrix} \gamma_t \\ \gamma_{t+1} \end{pmatrix}}} \tag{149}$$

And the standard error of $\epsilon_{k,p}$ can be calculated as

$$\operatorname{se}(\epsilon_{k,p}) = \left(\pi_o(\pi_p - \pi_p^2)\frac{1}{\lambda_o} + \pi_p^2(\pi_o - \pi_o^2)\lambda_o IV_o \right) \cdot x_{k,p} \cdot \operatorname{se}(\beta_k) \tag{150}$$

and in case of a quadratic covariate $\bar{x}_{k,p}^2$

$$\frac{\partial \epsilon_{k,p}}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}} = \begin{pmatrix} \left(\pi_o(\pi_p - \pi_p^2)\frac{1}{\lambda_o} + \pi_p^2(\pi_o - \pi_o^2)\lambda_o IV_o \right) x_{k,p} \\ \left(\pi_o(\pi_p - \pi_p^2)\frac{1}{\lambda_o} + \pi_p^2(\pi_o - \pi_o^2)\lambda_o IV_o \right) 2 x_{k,p}^2 \end{pmatrix} \tag{151}$$

$$\operatorname{se}(\epsilon_{k,p}) = \sqrt{\left(\frac{\partial \epsilon_{k,p}}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}} \right)^\top \cdot \operatorname{Var} \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix} \cdot \frac{\partial \epsilon_{k,p}}{\partial \begin{pmatrix} \beta_k \\ \beta_{k+1} \end{pmatrix}}} \tag{152}$$

Using the helper functions `checkXPos` (checking argument `xPos`, see appendix A) and `check-PKXBeta` (checking $\beta_0 + \sum_{j=1}^{K} \beta_j x_j$ for plausible values, see appendix A), the following function calculates the semi-elasticities and their respective standard errors as defined in equations (120), (121), (123), and (125) for the binary logit function, the semi-elasticities and their respective approximate standard errors as defined in equations (127), (130), (131), and (133) for the multi-nomial logit function, the semi-elasticities and their respective approcimate standard errors as defined in equations (135), (136), (137), and (139) for the conditional logit

```r
logEla <- function( allCoef, allCoefBra, allXVal, allXValBra,
                    allCoefSE = rep( NA, length( allCoef ) ),
                    xPos, yCat, yCatBra, lambda, method =  "binary" ){
  if( method == "binary" || method == "CondL" ){
    nCoef <- length( allCoef )
    # Checking standard errors
    if( nCoef != length( allCoefSE ) ) {
      stop( "arguments 'allCoef' and 'allCoefSE' must have the same length" )
    }
  } else if( method == "MNL" ){
    NCoef <- length( allCoef )
    mCoef <- matrix( allCoef, nrow = length( allXVal ) )
    nCoef <- dim( mCoef )[1]
    pCoef <- dim( mCoef )[2]
    # Checking standard errors
    if( NCoef != length( allCoefSE ) ) {
      stop( "arguments 'allCoef' and 'allCoefSE' must have the same length" )
    }
  } else{
    nCoef <- length( allCoef )
    NCoef <- length( allCoefBra )
    # Checking standard errors
    if( nCoef != length( allCoefSE ) ){
      stop( "arguments 'allCoef' and 'allCoefSE' must have the same length" )
    }
  }
  # Check position vector
  checkXPos( xPos, minLength = 1, maxLength = 2, minVal = 1, maxVal = nCoef )
  # Check x values
  if( method == "binary" || method == "MNL" ){
    if( nCoef != length( allXVal ) ) {
      stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
    }
  } else if( method == "CondL" ){
    mXVal <- matrix( allXVal, nrow = nCoef )
    nXVal <- dim( mXVal )[1]
    pXVal <- dim( mXVal )[2]
    if( nCoef != dim( mXVal )[1] ) {
      stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
    }
  } else{
    mXValBra <- matrix( allXValBra, nrow = NCoef )
    nXValBra <- dim( mXValBra )[1]
    pXValBra <- dim( mXValBra )[2]
    if( NCoef != nXValBra ) {
      stop( "arguments 'allCoefBra' and 'allXValBra' must have the same length" )
```

```
    }
    O <- length( allXVal )
    mXVal <- matrix( unlist( allXVal[ yCatBra ] ), nrow = nCoef )
    nXVal <- dim( mXVal )[1]
    pXVal <- dim( mXVal )[2]
    if( nCoef != nXVal ) {
      stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
    }
  }
  # Identify coefficients of interest (kth/tth covariate)
  if( length( xPos ) == 2 ){
    if( method == "binary" ){
      xCoef <- allCoef[ xPos ]
      if( !isTRUE( all.equal( allXVal[xPos[2]], allXVal[xPos[1]]^2 ) ) ) {
        stop( "the value of 'allXVal[ xPos[2] ]' must be equal",
          "to the squared value of 'allXVal[ xPos[1] ]' " )
      }
    } else if( method == "MNL" ){
      xCoef <- mCoef[ xPos, ]
      if( !isTRUE( all.equal( allXVal[xPos[2]], allXVal[xPos[1]]^2 ) ) ) {
        stop( "the value of 'allXVal[ xPos[2] ]' must be equal",
          "to the squared value of 'allXVal[ xPos[1] ]' " )
      }
    } else if( method == "CondL" ){
      xCoef <- allCoef[ xPos ]
      for( p in 1:pXVal ){
        if( !isTRUE( all.equal( mXVal[xPos[2], p], mXVal[xPos[1], p]^2 ) ) ) {
          stop( "the value of 'allXVal[ xPos[2] ]' must be equal",
            "to the squared value of 'allXVal[ xPos[1] ]' " )
        }
      }
    } else{
      xCoef <- allCoef[ xPos ]
      for( p in 1:pXVal ){
        if( !isTRUE( all.equal( mXVal[xPos[2], p], mXVal[xPos[1], p]^2 ) ) ) {
          stop( "the value of 'allXVal[ xPos[2] ]' must be equal",
            "to the squared value of 'allXVal[ xPos[1] ]' " )
        }
      }
    }
  } else if( length( xPos ) == 1 ) {
    if( method == "binary" || method == "CondL" ){
      xCoef <- c( allCoef[ xPos ], 0 )
    } else if( method == "MNL" ){
      xCoef <- matrix( c( mCoef[ xPos, ], rep( 0, dim( mCoef )[ 2 ] ) ),
        nrow = 2, byrow = TRUE  )
```

```
  } else{
    xCoef <- c( allCoef[ xPos ], 0 )
  }
} else {
  stop( "argument 'xPos' must be a scalar or a vector with two elements" )
}
if( method == "binary" ){
  xVal <- allXVal[ xPos[1] ]
  xBeta <- sum( allCoef * allXVal )
  checkXBeta( xBeta )
  dfun <- exp( xBeta )/( 1 + exp( xBeta ) )^2
  semEla <- ( xCoef[ 1 ] + 2 * xCoef[ 2 ] * xVal ) * xVal * dfun
} else if( method == "MNL" ){     #checkXBeta missing
  xVal <- allXVal[ xPos[1] ]
  xBeta <- colSums( mCoef * allXVal )
  pfun <- rep( NA, length( xBeta ))
  term <- 0
  for( i in 1:length( xBeta )){
    pfun[i] <- exp( xBeta[i] )/( 1 + sum( exp( xBeta ) ) )
    term <- term + ( ( xCoef[ 1, yCat ] + 2 * xCoef[ 2, yCat ] * xVal ) -
        ( xCoef[ 1, i ] + 2 * xCoef[ 2, i ] * xVal ) * pfun[i] )
  }
  semEla <- xVal * pfun[ yCat ] *
    ( ( xCoef[ 1, yCat ] + 2 * xCoef[ 2, yCat ] * xVal )/
        ( 1 + sum( exp( xBeta ))) + term )
  dfun <- pfun[ yCat ] * ( 1/( 1 + sum( exp( xBeta ) ) ) + term )
} else if( method == "CondL" ){    #checkXBeta missing
  xVal <- rep( NA, pXVal )
  for( p in 1:pXVal ){
    xVal[p] <- mXVal[ xPos[ 1 ], p ]
  }
  xBeta <- colSums( allCoef * mXVal )
  pfun <- exp( xBeta[ yCat ] )/( sum( exp( xBeta ) ) )
  semEla <- ( xCoef[1] + 2 * xCoef[2] * xVal[ yCat ] ) *
    xVal[ yCat ] * ( pfun - pfun^2 )
} else{                           #checkXBeta missing
  xVal <- rep( NA, pXVal )
  for( p in 1:pXVal ){
    xVal[p] <- mXVal[ xPos[ 1 ], p ]
  }
  coef <- matrix( NA, nrow = O, ncol = nCoef )
  for( o in 1:O ){
    coef[o, ] <- allCoef/lambda[o]
  }
  xBeta <- lapply( 1:O, function( i, m, v ){ colSums( m[[i]] * v[[i]] ) },
    m=allXVal, v=coef )
```

```
    IV <- unlist( lapply( 1:O, function( i, m ){ log( sum( exp( m[[i]] ) ) ) },
      m=xBeta ) )
    pfun <- exp( xBeta[[ yCatBra ]][ yCat ] )/
      ( sum( exp( xBeta[[ yCatBra ]] ) ) )
    xBetaBra <- colSums( allCoefBra * mXValBra )
    pfunBra <- exp( xBetaBra[ yCatBra ] + lambda[ yCatBra ] * IV[ yCatBra ] )/
      ( sum( exp( xBetaBra + lambda * IV ) ) )
    semEla <- ( xCoef[1] + 2 * xCoef[2] * xVal[ yCat ] ) * xVal[ yCat ] *
      ( pfunBra * ( pfun - pfun^2 ) * 1/lambda[ yCatBra ] +
          pfun^2 * ( pfunBra - pfunBra^2 ) * lambda[ yCatBra ] * IV[ yCatBra ] )
  }
  if( method == "binary" || method == "MNL" ){
    derivCoef <- rep( 0, length( allCoef ) )
    derivCoef[ xPos[1] ] <- dfun * xVal
    if( length( xPos ) == 2 ) {
      derivCoef[ xPos[2] ] <- dfun * 2 * xVal^2
    }
  } else if( method == "CondL" ){
    derivCoef <- rep( 0, length( allCoef ) )
    derivCoef[ xPos[1] ] <- ( pfun - pfun^2 ) * xVal[ yCat ]
    if( length( xPos ) == 2 ) {
      derivCoef[ xPos[2] ] <- ( pfun - pfun^2 ) * 2 * xVal[ yCat ]^2
    }
  } else{
    derivCoef <- rep( 0, length( allCoef ) )
    derivCoef[ xPos[1] ] <- (
      pfunBra * ( pfun - pfun^2 ) / lambda[ yCatBra ] +
        pfun^2 * ( pfunBra - pfunBra^2 ) * lambda[ yCatBra ] *
        IV[ yCatBra ] ) *
      xVal[ yCat ]
    if( length( xPos ) == 2 ) {
      derivCoef[ xPos[2] ] <- (
        pfunBra * ( pfun - pfun^2 ) / lambda[ yCatBra ] +
          pfun^2 * ( pfunBra - pfunBra^2 ) * lambda[ yCatBra ] *
          IV[ yCatBra ] ) *
        2 * xVal[ yCat ]^2
    }
  }
  vcovCoef <- diag( allCoefSE^2 )
  semElaSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  result <- c( semEla = semEla, stdEr = semElaSE )
  return( result )
}
```

with

- arguments `allCoef` $= (\beta_0, \ldots, \beta_K)^\top$ a vector of all coefficients from a binary logit

model, `allCoef` $= (\gamma_0, \ldots, \gamma_T)^\top$ a vector of all coefficients from a conditional logit model, `allCoefBra` $= (\beta_0, \ldots, \beta_K)^\top$, a vector of all coefficients from a nested logit model, where `allCoefBra`, the coefficients at the branch level, must always be included in combination with `allCoef`, the coefficients at the twig level, or `allCoef` $= (\beta_{01}, \ldots, \beta_{K1}, \ldots, \beta_{0P}, \ldots, \beta_{KP})$ a vector of the P sets of coefficients from the multinomial logit regression which does **not** include any values for the reference category;

- argument `allXVal` $= (1, \ldots, \bar{x}_K)^\top$, a vector of all corresponding sample means and 1 for the intercept for the binary or multi-nomial logit, `allXVal` $= (1, \ldots, \bar{z}_{T,1}, \ldots, 1, \ldots, \bar{z}_{T,P})$ a vector of the P sets of explanatory variables from the conditional logit, or `allXVal` $= (((1, \ldots, \bar{x}_{K,p,o}), \ldots, (1, \ldots, \bar{x}_{K,P,o}))^\top, \ldots, ((1, \ldots, \bar{x}_{K,p,O}), \ldots, (1, \ldots, \bar{x}_{K,P,O}))^\top)^\top$, a list where the list elements are the corresponding matrices of the sample means for each nest at the twig level,[8] and which must always be combined with the corresponding values at the branch level in `allXVal`;

- argument `allCoefSE` $= (\text{se}(\beta_0), \ldots, \text{se}(\beta_K))^\top$, a vector of standard errors for all coefficients of a binary or conditional logit regression or of the standard errors of the P sets of coefficients in a multi-nomial logit regression;

- argument `xPos` $= k + 1$ or $(k + 1, k + 2)^\top$ a scalar or vector of two elements indicating the position of the coefficients of interest in vectors `allCoef` and `allXVal`;

- argument `method = c("binary", "MNL", "CondL", "NestL" )` indicating the estimator used, where option `"CondL"` can also be used to calculate the semi-elasticity and standard error of a nested logit at the branch level and where option `"NestL"` estimates the semi-elasticity and standard error of the combined probabilities at the branch and twig level;

- argument `"lambda"` is a vector of the O parameter of the correlation coefficient between the the branch specific alternatives of interest, it is only relevant under option `"NestL"` and should be set to 1 in case the estimation coefficients in `allCoef` are already corrected by $\frac{\beta_k}{\lambda_o}$;

- finally, `yCat` a scalar identifying which of the P output categories is of interest, only in combination with `method = "MNL"`, `"CondL"`, and `"NestL"` (for the twig level), and `yCatBra` at the branch level.

```
# Example
ela6a <- logEla( allCoef = c( 0.445, 0.03, 0.00002, 0.067, 0.89, 0.124 ),
                 allXVal = c( 1, 3.3, 4.5, 2.34, 0.1, 0.987 ), xPos = 2 )
ela6a

##     semEla      stdEr
## 0.02023896         NA
```

[8] The reason why we make an exception from the usual vector in the case of the nested logit model is that nests, o, can have different dimensions wrt P.

```
ela6b <- logEla( allCoef = c( 0.445, 0.03, 0.00002, 0.067, 0.89, 0.124 ),
                 allXVal = c( 1, 3.3, 4.5, 2.24, 0.1, 0.987 ),
                 allCoefSE = c( 0.001, 0.02, 0.000002, 0.05, 1.2, 0.03 ),
                 xPos = 2 )
ela6b

##     semEla      stdEr
## 0.02029675 0.01353117

# Example
ela7a <- logEla( allCoef = c( 0.445, 0.03, 0.00002, 0.067, 0.89, 0.124 ),
                 allXVal = c( 1, 3.3, 3.3^2, 2.34, 0.1, 0.987 ),
                 xPos = c( 2, 3 ) )
ela7a

##     semEla      stdEr
## 0.02032691         NA

ela7b <- logEla( allCoef = c( 0.445, 0.03, 0.00002, 0.067, 0.89, 0.124 ),
                 allXVal = c( 1, 3.3, 3.3^2, 2.34, 0.1, 0.987 ),
                 allCoefSE = c( 0.001, 0.02, 0.000002, 0.05, 1.2, 0.03 ),
                 xPos = c( 2, 3 ) )
ela7b

##     semEla      stdEr
## 0.02032691 0.01349191

# Example
ela8a <- logEla( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                 allXVal = c( 1, 8.4, 0.06 ), xPos = 3,
                 method = "MNL", yCat = 2 )
ela8a

##      semEla       stdEr
## 0.002792501          NA

ela8b <- logEla( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                 allXVal = c( 1, 8.4, 0.06 ),
                 allCoefSE = c( 0.002, 0.003, 0.004, 0.006, 0.00001, 0.08 ),
                 xPos = 3,
                 method = "MNL", yCat = 2 )
ela8b

##       semEla        stdEr
## 2.792501e-03 1.150331e-05
```

```r
# Example
ela9a <- logEla( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                 allXVal = c( 1, 0.04, 0.0016 ), xPos = c( 2, 3 ),
                 method = "MNL", yCat = 2 )
ela9a

##       semEla        stdEr
## 0.0002470181           NA

ela9b <- logEla( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                 allXVal = c( 1, 0.04, 0.0016 ),
                 allCoefSE = c( 0.002, 0.003, 0.004, 0.006, 0.00001, 0.08 ),
                 xPos = c( 2, 3 ),
                 method = "MNL", yCat = 2 )
ela9b

##       semEla        stdEr
## 2.470181e-04 1.051034e-05

# Example
ela10a <- logEla( allCoef = c( 0.445, 0.03, 0.00002 ),
                  allXVal = c( 1, 3.3, 4.5, 1, 0.1, 0.987 ),
                  xPos = 2,
                  method = "CondL", yCat = 2 )
ela10a

##       semEla        stdEr
## 0.0007482721           NA

ela10b <- logEla( allCoef = c( 0.445, 0.03, -0.002 ),
                  allXVal = c( 1, 0.3, 0.09, 1, 0.1, 0.01 ),
                  xPos = c( 2, 3 ),
                  method = "CondL", yCat = 2 )
ela10b

##       semEla        stdEr
## 0.0007399937           NA

# Example
ela11a <- logEla( allCoef = c( 0.445, 0.03, 0.00002 ),
                  allXVal = c( 1, 3.3, 4.5, 1, 0.1, 0.987 ),
                  allCoefSE = c( 0.002, 0.003, 0.004 ),
                  xPos = 2,
                  method = "CondL", yCat = 2 )
ela11a

##       semEla        stdEr
## 7.482721e-04 7.482721e-05
```

```r
ela11b <- logEla( allCoef = c( 0.445, 0.03, -0.002 ),
                  allXVal = c( 1, 0.3, 0.09, 1, 0.1, 0.01 ),
                  allCoefSE = c( 0.002, 0.003, 0.004 ),
                  xPos = c( 2, 3 ),
                  method = "CondL", yCat = 2 )
ela11b
```

```
##       semEla        stdEr
## 7.399937e-04 7.762021e-05
```

```r
# Example
matrix1 <- matrix( c( 1, 2.5, 0.3, 0.09, 1, 0.33, 0.9, 1.8 ), nrow = 4 )
matrix2 <- matrix( c( 1, 2.8, 0.099, 0.211 ), nrow = 4 )
ela12a <- logEla( allCoefBra = c( 0.445, 0.03, -0.002 ),
                  allCoef = c( 1.8, 0.005, -0.12, 0.8 ),
                  allXValBra = c( 1, 3.3, 4.5, 1, 0.1, 0.987 ),
                  allXVal = list( matrix1, matrix2 ),
                  xPos = 2, yCatBra = 1, yCat = 2, lambda = c( 0.8, 1 ),
                  method = "NestedL" )
ela12a
```

```
##      semEla       stdEr
## 0.001321595          NA
```

```r
matrix1 <- matrix( c( 1, 0.3, 0.09, 0.09, 1, 0.33, 0.1089, 1.8 ), nrow = 4 )
matrix2 <- matrix( c( 1, 0.31, 0.099, 0.211 ), nrow = 4 )
ela12b <- logEla( allCoefBra = c( 0.445, 0.03, -0.002 ),
                  allCoef = c( 1.8, 0.005, -0.12, 0.8 ),
                  allXValBra = c( 1, 3.3, 4.5, 1, 0.1, 0.987 ),
                  allXVal = list( matrix1, matrix2 ),
                  xPos = 2, yCatBra = 1, yCat = 2, lambda = c( 0.8, 1 ),
                  method = "NestedL" )
ela12b
```

```
##      semEla       stdEr
## 0.000448864          NA
```

```r
# Example
matrix1 <- matrix( c( 1, 2.5, 0.3, 0.09, 1, 0.33, 0.9, 1.8 ), nrow = 4 )
matrix2 <- matrix( c( 1, 2.8, 0.099, 0.211 ), nrow = 4 )
ela13a <- logEla( allCoefBra = c( 0.445, 0.03, -0.002 ),
                  allCoef = c( 1.8, 0.005, -0.12, 0.8 ),
                  allXValBra = c( 1, 3.3, 4.5, 1, 0.1, 0.987 ),
                  allXVal = list( matrix1, matrix2 ),
                  allCoefSE = c( 0.001, 0.089, 0.0003, 0.12 ),
                  xPos = 2, yCatBra = 1, yCat = 2, lambda = c( 0.8, 1 ),
                  method = "NestedL" )
ela13a
```

```
##      semEla       stdEr
## 0.001321595 0.023524389

matrix1 <- matrix( c( 1, 0.3, 0.09, 0.09, 1, 0.33, 0.1089, 1.8 ), nrow = 4 )
matrix2 <- matrix( c( 1, 0.31, 0.099, 0.211 ), nrow = 4 )
ela13b <- logEla( allCoefBra = c( 0.445, 0.03, -0.002 ),
                  allCoef = c( 1.8, 0.005, -0.12, 0.8 ),
                  allXValBra = c( 1, 3.3, 4.5, 1, 0.1, 0.987 ),
                  allXVal = list( matrix1, matrix2 ),
                  allCoefSE = c( 0.001, 0.089, 0.0003, 0.12 ),
                  xPos = 2, yCatBra = 1, yCat = 2, lambda = c( 0.8, 1 ),
                  method = "NestedL" )
ela13b

##      semEla       stdEr
## 0.000448864 0.007989780
```

3.2. Semi-elasticities from interval-coded explanatory variables

Logit model, where the kth explanatory variable is interval-coded:

$$\Pr(y=1|x) = \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\backslash k}\beta_j x_j + \sum_{m\in\{1,\dots,M\}\backslash m^*}\delta_m D_m\right)}{1+\exp\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\backslash k}\beta_j x_j + \sum_{m\in\{1,\dots,M\}\backslash m^*}\delta_m D_m\right)}, \tag{153}$$

with

$$D_m = \begin{cases} 1 & \text{if } b_{m-1} < x_k \leq b_m \\ 0 & \text{otherwise} \end{cases} \quad \forall\, m = 1,\dots,M, \tag{154}$$

The semi-elasticities from an interval coded variable in the case of a binary logit function can be calculated as in section 2.2, where equation (84) modifies to

$$\epsilon_{km} \approx 2\left(\frac{\exp_{m+1}(\cdot)}{1+\exp_{m+1}(\cdot)} - \frac{\exp_m(\cdot)}{1+\exp_m(\cdot)}\right)\cdot\frac{b_m}{b_{m+1}-b_{m-1}} \tag{155}$$

where

$$\exp_{m+1}(\cdot) \equiv \exp\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\backslash k}\beta_j x_j + \delta_{m+1}\right) \tag{156}$$

and

$$\exp_m(\cdot) \equiv \exp\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\backslash k}\beta_j x_j + \delta_m\right). \tag{157}$$

For the multi-nomial logit, equation (155) modifies to

$$\epsilon_{km,p} \approx 2\left(\frac{\exp_{m+1,p}(\cdot)}{1+\sum_{p\in\{1,\dots,P\}\backslash p^*}\exp_{m+1,p}(\cdot)} - \frac{\exp_{m,p}(\cdot)}{1+\sum_{p\in\{1,\dots,P\}\backslash p^*}\exp_{m,p}(\cdot)}\right)\cdot\frac{b_m}{b_{m+1}-b_{m-1}} \tag{158}$$

where $\exp_{m,p}$ is defined as

$$\exp_{m,p}(\cdot) \equiv \exp\left(\beta_{0,p} + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_{j,p} x_j + \delta_{m,p}\right). \tag{159}$$

and $\exp_{m+1,p}$ accordingly.

For the conditional logit, equation (155) modifies to

$$\epsilon_{tm,p} \approx 2\left(\frac{\exp_{m+1,p}(\cdot)}{\sum_{p\in C}\exp_{m+1,p}(\cdot)} - \frac{\exp_{m,p}(\cdot)}{\sum_{p\in C}\exp_{m,p}(\cdot)}\right)\cdot\frac{b_m}{b_{m+1}-b_{m-1}} \tag{160}$$

where $\exp_{m,p}$ is defined as

$$\exp_{m,p}(\cdot) \equiv \exp\left(\gamma_0 + \sum_{j\in\{1,\ldots,T\}\setminus t} \gamma_j z_{j,p} + \delta_m\right). \tag{161}$$

and $\exp_{m+1,p}$ accordingly.

An approximate standard error of the semi-elasticity of interval-coded explanatory variables of binary logit models can be obtained by using the Delta-method:

$$\operatorname{se}(\epsilon_{km}) = \sqrt{\frac{\partial\epsilon_{km}}{\partial\left(\beta^\top\ \delta^\top\right)}\operatorname{Var}\begin{pmatrix}\beta\\ \delta\end{pmatrix}\frac{\partial\epsilon_{km}}{\partial\begin{pmatrix}\beta\\ \delta\end{pmatrix}}}, \tag{162}$$

where $\operatorname{se}(\epsilon_{km})$ indicates the (approximate) standard error of ϵ_{km}, $\operatorname{Var}\left(\left(\beta^\top\ \delta^\top\right)^\top\right)$ indicates the variance-covariance matrix of the estimates of the coefficient vector $\left(\beta^\top\ \delta^\top\right)^\top$, and the gradient vector is:

$$\frac{\partial\epsilon_{km}}{\partial\begin{pmatrix}\beta\\ \delta\end{pmatrix}} = \sum_{m=1}^{M-1} w_m \frac{\partial\epsilon_{km}}{\partial\begin{pmatrix}\beta\\ \delta\end{pmatrix}} \tag{163}$$

with the elements of the gradient vector $\partial\epsilon_{km}/\partial(\beta^\top\delta^\top)^\top$:

$$\frac{\partial\epsilon_{km}}{\partial\beta_0} = 2\left(\frac{\exp_{m+1}(\cdot)}{[1+\exp_{m+1}(\cdot)]^2} - \frac{\exp_m(\cdot)}{[1+\exp_m(\cdot)]^2}\right)\cdot\frac{b_m}{b_{m+1}-b_{m-1}} \tag{164}$$

$$\frac{\partial\epsilon_{km}}{\partial\beta_j} = 2\left(\frac{\exp_{m+1}(\cdot)x_j}{[1+\exp_{m+1}(\cdot)]^2} - \frac{\exp_m(\cdot)x_j}{[1+\exp_m(\cdot)]^2}\right)\cdot\frac{b_m}{b_{m+1}-b_{m-1}}\ \forall\ j\in\{1,\ldots,K\}\setminus k \tag{165}$$

$$\frac{\partial\epsilon_{km}}{\partial\delta_m} = -2\frac{\exp_m(\cdot)}{[1+\exp_m(\cdot)]^2}\cdot\frac{b_m}{b_{m+1}-b_{m-1}} \tag{166}$$

$$\frac{\partial\epsilon_{km}}{\partial\delta_{m+1}} = 2\frac{\exp_{m+1}(\cdot)}{[1+\exp_{m+1}(\cdot)]^2}\cdot\frac{b_m}{b_{m+1}-b_{m-1}} \tag{167}$$

$$\frac{\partial\epsilon_{km}}{\partial\delta_n} = 0\ \forall\ n\in\{1,\ldots,M\}\setminus\{m,m+1\} \tag{168}$$

In the case of the multi-nomial logit the elements of the gradient vector $\partial\epsilon_{km,p}/\partial(\beta^\top\delta^\top)^\top$ modify to

$$\begin{aligned}\frac{\partial\epsilon_{km,p}}{\partial\beta_{0,p}} =& 2\left(\frac{\exp_{m+1,p}(\cdot)\cdot(1+\sum_{o\in\{1,\dots,P\}\setminus p,p^*}\exp_{m+1,o}(\cdot))}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m+1,p}(\cdot)]^2}\right. \\ &\left.-\frac{\exp_{m,p}(\cdot)\cdot(1+\sum_{o\in\{1,\dots,P\}\setminus p,p^*}\exp_{m,o}(\cdot))}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m,p}(\cdot)]^2}\right)\cdot\frac{b_m}{b_{m+1}-b_{m-1}}\end{aligned} \tag{169}$$

$$\begin{aligned}\frac{\partial\epsilon_{km,p}}{\partial\beta_{0,o}} =& 2\left(\frac{\exp_{m,p}(\cdot)\exp_{m,o}(\cdot)}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m,p}(\cdot)]^2}-\frac{\exp_{m+1,p}(\cdot)\exp_{m+1,o}(\cdot)}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m+1,p}(\cdot)]^2}\right) \\ &\cdot\frac{b_m}{b_{m+1}-b_{m-1}}\quad\forall\, o\neq p\end{aligned} \tag{170}$$

$$\begin{aligned}\frac{\partial\epsilon_{km,p}}{\partial\beta_{j,p}} =& 2\left(\frac{\exp_{m+1,p}(\cdot)\cdot x_j(1+\sum_{o\in\{1,\dots,P\}\setminus p,p^*}\exp_{m+1,o}(\cdot))}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m+1,p}(\cdot)]^2}\right. \\ &\left.-\frac{\exp_{m,p}(\cdot)\cdot x_j(1+\sum_{o\in\{1,\dots,P\}\setminus p,p^*}\exp_{m,o}(\cdot))}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m,p}(\cdot)]^2}\right)\cdot\frac{b_m}{b_{m+1}-b_{m-1}}\quad\forall\, j\in\{1,\dots,K\}\setminus k\end{aligned} \tag{171}$$

$$\begin{aligned}\frac{\partial\epsilon_{km,p}}{\partial\beta_{j,o}} =& 2\left(\frac{\exp_{m,p}(\cdot)\exp_{m,o}(\cdot)\cdot x_j}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m,p}(\cdot)]^2}-\frac{\exp_{m+1,p}(\cdot)\exp_{m+1,o}(\cdot)\cdot x_j}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m+1,p}(\cdot)]^2}\right) \\ &\cdot\frac{b_m}{b_{m+1}-b_{m-1}}\quad\forall\, o\neq p,\ j\in\{1,\dots,K\}\setminus k\end{aligned} \tag{172}$$

$$\frac{\partial\epsilon_{km}}{\partial\delta_{m,p}} = -2\frac{\exp_{m,p}(\cdot)\cdot(1+\sum_{o\in\{1,\dots,P\}\setminus p,p^*}\exp_{m,o}(\cdot))}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m,p}(\cdot)]^2}\cdot\frac{b_m}{b_{m+1}-b_{m-1}} \tag{173}$$

$$\frac{\partial\epsilon_{km,p}}{\partial\delta_{m,o}} = 2\frac{\exp_{m,p}(\cdot)\exp_{m,o}(\cdot)}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m,p}(\cdot)]^2}\cdot\frac{b_m}{b_{m+1}-b_{m-1}}\quad\forall\, o\neq p \tag{174}$$

$$\frac{\partial\epsilon_{km,p}}{\partial\delta_{m+1,p}} = 2\frac{\exp_{m+1,p}(\cdot)\cdot(1+\sum_{o\in\{1,\dots,P\}\setminus p,p^*}\exp_{m+1,o}(\cdot))}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m+1,p}(\cdot)]^2}\cdot\frac{b_m}{b_{m+1}-b_{m-1}} \tag{175}$$

$$\frac{\partial\epsilon_{km,p}}{\partial\delta_{m+1,o}} = -2\frac{\exp_{m+1,p}(\cdot)\exp_{m+1,o}(\cdot)}{[1+\sum_{p\in\{1,\dots,P\}\setminus p^*}\exp_{m+1,p}(\cdot)]^2}\cdot\frac{b_m}{b_{m+1}-b_{m-1}}\quad\forall\, o\neq p \tag{176}$$

$$\frac{\partial\epsilon_{km,p}}{\partial\delta_{n,p}} = 0\ \ \forall\, n\in\{1,\dots,M\}\setminus\{m,m+1\} \tag{177}$$

$$\frac{\partial\epsilon_{km,p}}{\partial\delta_{n,o}} = 0\ \ \forall\, n\in\{1,\dots,M\}\setminus\{m,m+1\},\ o\neq p \tag{178}$$

In the case of the conditional logit the elements of the gradient vector $\partial\epsilon_{tm,p}/\partial(\gamma^\top\delta^\top)^\top$

modify to

$$\frac{\partial \epsilon_{tm}}{\partial \gamma_0} = 2 \left(\frac{\exp_{m+1,p}(\cdot) \cdot \sum_{p \in C} \exp_{m+1,p}(\cdot) - \exp_{m+1,p}(\cdot) \cdot \sum_{p \in C} \exp_{m+1,p}(\cdot)}{\left(\sum_{p \in C} \exp_{m+1,p}(\cdot) \right)^2} - \right. \tag{179}$$

$$\left. - \frac{\exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot) - \exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot)}{\left(\sum_{p \in C} \exp_{m,p}(\cdot) \right)^2} \right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}} = 0$$

$$\frac{\partial \epsilon_{tm}}{\partial \gamma_j} = 2 \left(\frac{\exp_{m+1,p}(\cdot) z_{j,p} \cdot \sum_{p \in C} \exp_{m+1,p}(\cdot) - \exp_{m+1,p}(\cdot) \cdot \sum_{p \in C} \exp_{m+1,p}(\cdot) z_{j,p}}{\left(\sum_{p \in C} \exp_{m+1,p}(\cdot) \right)^2} \right. \tag{180}$$

$$\left. - \frac{\exp_{m,p}(\cdot) z_{j,p} \cdot \sum_{p \in C} \exp_{m,p}(\cdot) - \exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot) z_{j,p}}{\left(\sum_{p \in C} \exp_{m,p}(\cdot) \right)^2} \right)$$

$$\cdot \frac{b_m}{b_{m+1} - b_{m-1}} \; \forall \; j \in \{1, \ldots, T\} \setminus t$$

$$\frac{\partial \epsilon_{tm}}{\partial \delta_m} = -2 \frac{\exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot) - \exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot)}{\left(\sum_{p \in C} \exp_{m,p}(\cdot) \right)^2} \tag{181}$$

$$\cdot \frac{b_m}{b_{m+1} - b_{m-1}} = 0$$

$$\frac{\partial \epsilon_{tm}}{\partial \delta_{m+1}} = 2 \frac{\exp_{m+1,p}(\cdot) \cdot \sum_{p \in C} \exp_{m+1,p}(\cdot) - \exp_{m+1,p}(\cdot) \cdot \sum_{p \in C} \exp_{m+1,p}(\cdot)}{\left(\sum_{p \in C} \exp_{m+1,p}(\cdot) \right)^2} \tag{182}$$

$$\cdot \frac{b_m}{b_{m+1} - b_{m-1}} = 0$$

$$\frac{\partial \epsilon_{km}}{\partial \delta_n} = 0 \; \forall \; n \in \{1, \ldots, M\} \setminus \{m, m+1\} \tag{183}$$

Using the helper functions `elaIntWeights` (equation 21), `elaIntBounds` (equation 20), `checkXPos` (checking argument `xPos`, see appendix A), and `checkXBeta` (checking $\beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \sum_{m \in \{1,\ldots,M\} \setminus m^*} \delta_m D_m$ for plausible values, see appendix A), and function `linElaInt` (section 1.2), the following code defines a function that calculates the semi-elasticity defined in equation (155) and its approximate standard error defined in equations (162) to (168) for the binary logit function, equations (158) and (170) to (178) for the multi-nomial logit function, and equations (160) and (179) to (183) for the conditional logit, respectively:

```
logElaInt <- function( allCoef, allXVal, xPos, xBound, yCat = NA,
                       allCoefSE = rep( NA, length( allCoef ) ),
```

```
                        method = "binary" ){
# number of coefficients
if( method == "binary" || method == "MNL" ){
  mCoef <- matrix( allCoef, nrow = length( allXVal ))
  nCoef <- dim( mCoef )[1]
  pCoef <- dim( mCoef )[2]
  # checking arguments
  if( length( allXVal ) != nCoef ) {
    stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
  }
} else{
  nCoef <- length( allCoef )
  mXVal <- matrix( allXVal, nrow = nCoef )
  pCoef <- dim( mXVal )[2]
  # checking arguments
  if( dim( mXVal )[1] != nCoef ) {
    stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
  }
}
# number of intervals
nInt <- length( xPos )
checkXPos( xPos, minLength = 2, maxLength = nCoef,
           minVal = 0, maxVal = nCoef, requiredVal = 0 )
if( method == "binary" || method == "MNL" ){
  if( any( allXVal[ xPos ] < 0 ) ) {
      stop( "all elements of argument 'allXVal'",
            " that are indicated by argument 'xPos'",
            " (i.e., the shares of observations in each interval)",
            " must be non-negative" )
    }
  if( sum( allXVal[ xPos ] > 1 ) ) {
      stop( "the sum of the elements of argument 'allXVal'",
            " that are indicated by argument 'xPos'",
            " (i.e., the shares of observations in each interval)",
            " must not be larger than one" )
  }
} else{
  for( p in 1:pCoef ){
  if( any( mXVal[ xPos, p ] < 0 ) ) {
      stop( "all elements of argument 'allXVal'",
            " that are indicated by argument 'xPos'",
            " (i.e., the shares of observations in each interval)",
            " must be non-negative" )
    }
  if( sum( mXVal[ xPos, p ] > 1 ) ) {
      stop( "the sum of the elements of argument 'allXVal'",
```

```
            " that are indicated by argument 'xPos'",
            " (i.e., the shares of observations in each interval)",
            " must not be larger than one" )
    }
  }
}
# check 'xBound' and replace infinite values
xBound <- elaIntBounds( xBound, nInt )
# vector of probabilities of y=1 for each interval and
# vector of shares of observations in each interval
xBeta <- matrix( rep( rep( NA, nInt ), pCoef ), ncol = pCoef )
if( method == "binary" || method == "MNL" ){
  shareVec <- rep( NA, nInt )
  for( p in 1:pCoef ){
        for( i in 1:nInt ){
            allXValTemp <- replace( allXVal, xPos, 0 )
            if( xPos[i] != 0 ) {
              allXValTemp[ xPos[i] ] <- 1
              shareVec[i] <- allXVal[ xPos[i] ]
            }
          xBeta[i,p] <- sum( mCoef[ ,p] * allXValTemp )
        }
  }
  shareVec[ xPos == 0 ] <- 1 - sum( shareVec[ xPos != 0 ] )
} else{
  shareVec <- matrix( rep( rep( NA, nInt ), pCoef ), ncol = pCoef )
  for( p in 1:pCoef ){
       for( i in 1:nInt ){
           allXValTemp <- replace( mXVal[ ,p], xPos, 0 )
           if( xPos[i] != 0 ) {
              allXValTemp[ xPos[i] ] <- 1
              shareVec[i,p] <- mXVal[ xPos[i], p ]
           }
         xBeta[i,p] <- sum( allCoef * allXValTemp )
       }
    shareVec[ xPos == 0, p ] <- 1 - sum( shareVec[ xPos != 0, p ] )
  }
  shareVec <- shareVec[ , yCat ]
}
#checkXBeta( xBeta )  #Please check this one with a matrix
if( method == "binary" ){
   expVec <- as.vector( exp( xBeta )/( 1 + exp( xBeta ) ) )
} else if( method == "MNL" ){
   expVec <- as.vector( exp( xBeta[ , yCat ])/( 1 + rowSums( exp( xBeta ) ) ) )
} else{
   expVec <- as.vector( exp( xBeta[ , yCat ])/( rowSums( exp( xBeta ) ) ) )
```

```
  }
  # weights
  weights <- elaIntWeights( shareVec )
  # calculation of the semi-elasticity
  semEla <- linElaInt( expVec, shareVec, xBound )
  ### calculation of its standard error
  # partial derivatives of each semi-elasticity around each boundary
  # w.r.t. all estimated coefficients
  if( method == "binary" ){
    gradM <- matrix( 0, nCoef, nInt - 1 )
    gradExpVec <- exp( xBeta )/( 1 + exp( xBeta ) )^2
    for( m in 1:( nInt - 1 ) ) {
      gradM[ -xPos, m ] <- 2 * ( gradExpVec[m+1] - gradExpVec[m] ) *
          allXVal[ -xPos ] * xBound[m+1] / ( xBound[m+2] - xBound[m] )
      gradM[ xPos[m], m ] <- - 2 * gradExpVec[m] * xBound[m+1] /
        ( xBound[m+2] - xBound[m] )
      gradM[ xPos[m+1], m ] <- 2 * gradExpVec[m+1] * xBound[m+1] /
        ( xBound[m+2] - xBound[m] )
    }
  } else if( method == "MNL" ){
    gradM <- array( 0, c( nCoef, nInt - 1, pCoef ) )
    gradExpVecP <- ( exp( xBeta[ , yCat ] ) *
         ( 1 + rowSums( exp( xBeta[ , -yCat, drop = FALSE ] ) ) ) )/
         ( 1 + rowSums( exp( xBeta ) ) )^2
    for( p in 1:pCoef ){
      gradExpVec0 <- ( exp( xBeta[ , yCat ] ) * exp( xBeta[ , p] ) )/
         ( 1 + rowSums( exp( xBeta ) ) )^2
      for( m in 1:( nInt - 1 ) ) {
        if( p == yCat ){
          gradM[ -xPos, m, p ] <- 2 * ( gradExpVecP[m+1] - gradExpVecP[m] ) *
               allXVal[ -xPos ] * xBound[m+1] / ( xBound[m+2] - xBound[m] )
          gradM[ xPos[ m ], m, p ] <- - 2 * gradExpVecP[m] * xBound[m+1] /
                                              ( xBound[m+2] - xBound[m] )
          gradM[ xPos[ m + 1 ], m, p ] <- 2 * gradExpVecP[m+1] * xBound[m+1] /
                                              ( xBound[m+2] - xBound[m] )
        } else {
          gradM[ -xPos, m, p ] <- 2 * ( gradExpVec0[m] - gradExpVec0[m+1] ) *
               allXVal[ -xPos ] * xBound[m+1] / ( xBound[m+2] - xBound[m] )
          gradM[ xPos[ m ], m, p ] <- 2 * gradExpVec0[m] * xBound[m+1] /
                                              ( xBound[m+2] - xBound[m] )
          gradM[ xPos[ m + 1 ], m, p ] <- - 2 * gradExpVec0[m+1] * xBound[m+1] /
                                              ( xBound[m+2] - xBound[m] )
        }
      }
    }
    gradM <- apply( gradM, 2, function( x ) x )
```

```
  } else{
    gradM <- matrix( 0, nCoef, nInt - 1 )
    for( m in 1:( nInt - 1 ) ) {
      gradM[ -xPos, m ] <- 2 *
          ( ( exp( xBeta[ m+1, yCat ] ) * mXVal[ -xPos, yCat ] *
              sum( exp( xBeta[ m+1, ] ) ) -
              exp( xBeta[ m+1, yCat ] ) *
              rowSums( exp( xBeta[ m+1, ] ) * mXVal[ -xPos, , drop = FALSE ] ) )/
            ( sum( exp( xBeta[ m+1, ] ) ) )^2 -
            ( exp( xBeta[ m, yCat ] ) * mXVal[ -xPos, yCat ] *
              sum( exp( xBeta[ m, ] ) ) -
              exp( xBeta[ m, yCat ] ) *
              rowSums( exp( xBeta[ m,  ] ) * mXVal[ -xPos, , drop = FALSE ] ) )/
            ( sum( exp( xBeta[ m, ] ) ) )^2 ) *
            xBound[m+1] / ( xBound[m+2] - xBound[m] )
      gradM[ xPos[m], m ] <- 0
      gradM[ xPos[m+1], m ] <- 0
    }
  }
  # partial derivative of the semi-elasticity
  # w.r.t. all estimated coefficients
  derivCoef <- rep( 0, length( allCoef ) )
  for( m in 1:( nInt - 1 ) ){
    derivCoef <- derivCoef + weights[m] * gradM[,m]
  }
  # variance-covariance matrix of the coefficiencts
  vcovCoef <- diag( allCoefSE^2 )
  # standard error of the (average) semi-elasticity
  semElaSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  # prepare object that will be returned
  result <- c( semEla[1], stdEr = semElaSE )
  return( result )
}
```

where argument `allCoef` $= (\beta_0, \ldots, \beta_{k-1}, \delta_1, \ldots, \delta_{m^*-1}, \delta_{m^*+1}, \ldots, \delta_M, \beta_{k+1}, \ldots, \beta_K)^\top$, is a vector of all coefficients from the binary or conditional logit regression or `allCoef` $= (\beta_{0,1}, \ldots, \beta_{k-1,1}, \delta_{1,1}, \ldots, \delta_{m^*-1,1}, \delta_{m^*+1,1}, \ldots, \delta_{M,1}, \beta_{k+1,1}, \ldots, \beta_{K,1}, \ldots,$ $\beta_{0,P}, \ldots, \beta_{k-1,P}, \delta_{1,P}, \ldots, \delta_{m^*-1,P}, \delta_{m^*+1,P}, \ldots, \delta_{M,P}, \beta_{k+1,P}, \ldots, \beta_{K,P})^\top$ a vector of all the P sets of coefficients from the multi-nomial logit regression which does **not** include any values for the reference category; argument `allXVal` $= (1, x_1, \ldots, x_{k-1}, s_1, \ldots, s_{m^*-1}, s_{m^*+1}, \ldots, s_M, x_{k+1}, \ldots, x_K)^\top$ is a vector of corresponding values of the explanatory variables (including shares of observations in each interval of variable x_k except for the reference interval m^*) or `allXVal` $= (1, z_{1,1}, \ldots, z_{1,T}, \ldots, 1, z_{1,P}, \ldots, z_{T,P})$ is a vector of the P sets of explanatory variables from the conditional logit (including shares of observations in each interval of variable $z_{t,p}$ except for the reference interval m^*); argument `xPos` $= (k+1, \ldots, k+m^*-1, 0, k+m^*, \ldots, k+M-1)$ is a vector indicating the positions of the coefficients $\delta_1, \ldots, \delta_M$ and shares $s_1, \ldots, s_M$ of each

interval in the vectors `allCoef` and `allXVal`, whereas the position of the reference interval m^* is set to zero; argument `allCoefSE` is a vector of the standard errors of all coefficients in `allCoef` or of the P sets of the standard errors of the coefficients from the multi-nomial logit regression which does **not** include any values for the reference category; argument `method = "binary", "MNL", "CondL"` identifies the estimator chosen, where `"binary"` indicates the binary logit estimator, `"MNL"` indicates the multi-nomial logit estimator, and `"CondL"` indicates the conditional logit estimator; argument `yCat`, only in combination with option `"MNL"` or `"CondL"`, indicates the pth output category of interest; and argument `xBound` $= (b_0, \ldots, b_M)^\top$ indicates the boundaries of the intervals as in function `linElaInt`.

```
# Example
ela8a <- logElaInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                    allXVal = c( 1, 0.4, 0.12, 0.13 ),
                    xPos = c( 2, 0, 3, 4 ),
                    xBound = c( 0, 500, 1000, 1500, Inf ) )
ela8a

##      semEla       stdEr
## -0.02442287          NA

ela8b <- logElaInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                    allXVal = c( 1, 0.4, 0.12, 0.13 ),
                    xPos = c( 2, 0, 3, 4 ),
                    xBound = c( 0, 500, 1000, 1500, Inf ),
                    allCoefSE = c( 0.003, 0.045, 0.007, 0.009 ) )
ela8b

##       semEla        stdEr
## -0.024422872  0.006055436

# Example
ela9a <- logElaInt( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                    allXVal = c( 1, 0.4, 0.12 ),
                    xPos = c( 2, 0, 3 ),
                    xBound = c( 0, 500, 1000, Inf ), yCat = 2,
                    method = "MNL" )
ela9a

##     semEla      stdEr
## 0.01775033         NA

ela9b <- logElaInt( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                    allXVal = c( 1, 0.4, 0.12 ),
                    xPos = c( 2, 0, 3 ),
                    xBound = c( 0, 500, 1000, Inf ), yCat = 2,
                    allCoefSE = c( 0.003, 0.045, 0.007, 0.009, 0.0008, 0.9 ),
                    method = "MNL" )
ela9b
```

```
##      semEla       stdEr
## 0.017750330 0.003286756

# Example
ela10a <- logElaInt( allCoef = c( 1.33, 0.022, 0.58, 1.6 ),
                     allXVal = c( 1, 0.4, 0.12, 0.0002,
                                  1, 0.28, 0.01, 0.000013 ),
                     xPos = c( 2, 0, 3 ),
                     xBound = c( 0, 1000, 1500, Inf ), yCat = 2,
                     method = "CondL" )
ela10a

##        semEla         stdEr
## -4.699944e-17            NA

ela10b <- logElaInt( allCoef = c( 1.33, 0.022, 0.58, 1.6 ),
                     allXVal = c( 1, 0.4, 0.12, 0.0002,
                                  1, 0.28, 0.01, 0.000013 ),
                     xPos = c( 2, 0, 3 ),
                     xBound = c( 0, 1000, 1500, Inf ),
                     allCoefSE = c( 0.003, 0.045, 0.007, 0.009 ), yCat = 2,
                     method = "CondL" )
ela10b

##        semEla         stdEr
## -4.699944e-17  3.838242e-20
```

3.3. Effects of continuous variables when they change between discrete intervals

As in section 3.1, we assume the following model

$$\Pr(y=1|x) = \frac{\exp\left(\beta_0 + \sum_{k=1}^{K} \beta_k x_k\right)}{1 + \exp\left(\beta_0 + \sum_{k=1}^{K} \beta_k x_k\right)}, \tag{184}$$

where $y \in \{0,1\}$ is a binary dependent variable, $x = (x_1, \ldots, x_K)^\top$ is a vector of K continuous explanatory variables, and $\beta = (\beta_0, \ldots, \beta_K)^\top$ is a vector of $K+1$ unknown coefficients.

In order to derive the (approximate) effect of a continuous variable x_k on $Pr(y=1)$ given that x_k changes between $M \geq 2$ intervals, we modify equation (31) into

$$\begin{aligned} E_{k,ml} =& E[y|b_{m-1} < x_k \leq b_m] - E[y|b_{l-1} < x_k \leq b_l] \\ \approx& \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \beta_k E[x_k|b_{m-1} < x_k \leq b_m]\right)}{1 + \exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \beta_k E[x_k|b_{m-1} < x_k \leq b_m]\right)} \\ & - \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \beta_k E[x_k|b_{l-1} < x_k \leq b_l]\right)}{1 + \exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \beta_k E[x_k|b_{l-1} < x_k \leq b_l]\right)} \end{aligned} \tag{185}$$

where again $E[x_k|b_{m-1} < x_k \leq b_m] = \bar{x}_{km}$ can be approximated by the mid-points of the intervals

$$\bar{x}_{km} \approx \frac{b_{m-1} + b_m}{2} \; \forall \; m = 1, \ldots, M. \tag{186}$$

For model specifications that additionally include a quadratic term of the explanatory variable,

$$\begin{aligned} E_{k,ml} =& E[y|b_{m-1} < x_k \leq b_m] - E[y|b_{l-1} < x_k \leq b_l] \\ \approx& \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus(k,k+1)} \beta_j x_j + \beta_k E[x_k|b_{m-1} < x_k \leq b_m] + \beta_{k+1} E[x_k^2|b_{m-1} < x_k \leq b_m]\right)}{1 + \exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus(k,k+1)} \beta_j x_j + \beta_k E[x_k|b_{m-1} < x_k \leq b_m] + \beta_{k+1} E[x_k^2|b_{m-1} < x_k \leq b_m]\right)} \\ & - \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus(k,k+1)} \beta_j x_j + \beta_k E[x_k|b_{l-1} < x_k \leq b_l] + \beta_{k+1} E[x_k^2|b_{l-1} < x_k \leq b_l]\right)}{1 + \exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus(k,k+1)} \beta_j x_j + \beta_k E[x_k|b_{l-1} < x_k \leq b_l] + \beta_{k+1} E[x_k^2|b_{l-1} < x_k \leq b_l]\right)} \end{aligned} \tag{187}$$

where the values for $E[x_k|b_{m-1} < x_k \leq b_m]$ and $E[x_k^2|b_{m-1} < x_k \leq b_m]$ remain the same as outlined in section 1.3.

In the case of the multi-nomial logit function equation (185) modifies to

$$\begin{aligned} E_{k,ml,p} \approx& \frac{\exp\left(\beta_{0,p} + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_{j,p} x_j + \beta_{k,p} E[x_k|b_{m-1} < x_k \leq b_m]\right)}{1 + \sum_{p\in\{1,\ldots,P\}\setminus p^*} \exp\left(\beta_{0,p} + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_{j,p} x_j + \beta_{k,p} E[x_k|b_{m-1} < x_k \leq b_m]\right)} \\ & - \frac{\exp\left(\beta_{0,p} + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_{j,p} x_j + \beta_{k,p} E[x_k|b_{l-1} < x_k \leq b_l]\right)}{1 + \sum_{p\in\{1,\ldots,P\}\setminus p^*} \exp\left(\beta_{0,p} + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_{j,p} x_j + \beta_{k,p} E[x_k|b_{l-1} < x_k \leq b_l]\right)} \end{aligned} \tag{188}$$

and equation (187) modifies accordingly.

In the case of the conditional logit function equation (185) modifies to

$$\begin{aligned} E_{k,ml,p} \approx& \frac{\exp\left(\gamma_0 + \sum_{j\in\{1,\ldots,T\}\setminus t} \gamma_j z_{j,p} + \gamma_k E[z_{k,p}|b_{m-1} < z_{k,p} \leq b_m]\right)}{\sum_{p\in C} \exp\left(\gamma_0 + \sum_{j\in\{1,\ldots,T\}\setminus t} \gamma_j z_{j,p} + \gamma_k E[z_{k,p}|b_{m-1} < z_{k,p} \leq b_m]\right)} \\ & - \frac{\exp\left(\gamma_0 + \sum_{j\in\{1,\ldots,T\}\setminus t} \gamma_j z_{j,p} + \gamma_k E[z_{k,p}|b_{l-1} < z_{k,p} \leq b_l]\right)}{\sum_{p\in C} \exp\left(\gamma_0 + \sum_{j\in\{1,\ldots,T\}\setminus t} \gamma_j z_{j,p} + \gamma_k E[z_{k,p}|b_{l-1} < z_{k,p} \leq b_l]\right)} \end{aligned} \tag{189}$$

and equation (187) modifies accordingly.

And in the case of the nested logit function equation (185) modifies to

$$
\begin{aligned}
E_{k,ml,p,o} = & \frac{\exp(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_o E[x_{k,p}|b_{m-1} < x_{k,p} \leq b_m])}{\sum_{p\in B_o} \exp(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p}) + \beta_k/\lambda_o E[x_{k,p}|b_{m-1} < x_{k,p} \leq b_m])} \\
& \cdot \frac{\exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o)}{\sum_{o=1}^{O} \exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o)} \\
& - \frac{\exp(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_o E[x_{k,p}|b_{l-1} < x_{k,p} \leq b_l])}{\sum_{p\in B_o} \exp(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p}) + \beta_k/\lambda_o E[x_{k,p}|b_{l-1} < x_{k,p} \leq b_l])} \\
& \cdot \frac{\exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o)}{\sum_{o=1}^{O} \exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o)}
\end{aligned} \tag{190}
$$

where equation (141) modifies to

$$
IV_o = \ln \sum_{p\in B_o} \exp\left(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_0 E[x_{k,p}|b_{m-1} < x_{k,p} \leq b_m]\right) \tag{191}
$$

and

$$
IV_o = \ln \sum_{p\in B_o} \exp\left(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_0 E[x_{k,p}|b_{l-1} < x_{k,p} \leq b_l]\right), \tag{192}
$$

respectively.

In order to calculate the standard error of $E_{k,ml}$, we again apply a simplified Delta method

$$
\operatorname{se}(E_{k,lm}) = \sqrt{\left(\frac{\partial E_{k,lm}}{\partial \beta}\right)^{\top} \cdot \operatorname{Var}(\beta) \cdot \frac{\partial E_{k,lm}}{\partial \beta}} \tag{193}
$$

where the elements of the gradient vector, $\dfrac{\partial E_{k,ml}}{\partial \beta}$, are:

$$
\frac{\partial E_{k,ml}}{\partial \beta_0} = \frac{\exp_m(\cdot)}{[1+\exp_m(\cdot)]^2} - \frac{\exp_l(\cdot)}{[1+\exp_l(\cdot)]^2} \tag{194}
$$

$$
\frac{\partial E_{k,ml}}{\partial \beta_j} = \frac{\exp_m(\cdot)\cdot \bar{x}_j}{[1+\exp_m(\cdot)]^2} - \frac{\exp_l(\cdot)\cdot \bar{x}_j}{[1+\exp_l(\cdot)]^2} \quad \forall \ \beta_j \ \ j \neq k, k+1 \tag{195}
$$

$$
\frac{\partial E_{k,ml}}{\partial \beta_k} = \frac{\exp_m(\cdot)\cdot \bar{x}_{km}}{[1+\exp_m(\cdot)]^2} - \frac{\exp_l(\cdot)\cdot \bar{x}_{kl}}{[1+\exp_l(\cdot)]^2} \tag{196}
$$

$$
\frac{\partial E_{k,ml}}{\partial \beta_{k+1}} = \frac{\exp_m(\cdot)\cdot \overline{x^2}_{km}}{[1+\exp_m(\cdot)]^2} - \frac{\exp_l(\cdot)\cdot \overline{x^2}_{kl}}{[1+\exp_l(\cdot)]^2} \tag{197}
$$

where

$$
\exp_m(\cdot) \equiv \exp\left(\beta_0 + \sum_{j\in\{1,\dots,K\}\setminus(k,k+1)} \beta_j x_j + \beta_k E[x_k|b_{m-1} < x_k \leq b_m] + \beta_{k+1} E[x_k^2|b_{m-1} < x_k \leq b_m]\right)
$$

and

$$\exp_l(\cdot) \equiv \exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus(k,k+1)} \beta_j x_j + \beta_k E[x_k | b_{l-1} < x_k \leq b_l] + \beta_{k+1} E[x_k^2 | b_{l-1} < x_k \leq b_l]\right).$$

In the case of the multi-nomial logit the gradient vector, $\frac{\partial E_{k,ml,p}}{\partial \beta}$, modifies to:

$$\frac{\partial E_{k,ml,p}}{\partial \beta_{0,p}} = \frac{\exp_{m,p}(\cdot) \cdot (1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{m,o}(\cdot))}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{m,p}(\cdot)]^2} - \frac{\exp_{l,p}(\cdot) \cdot (1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{l,o}(\cdot))}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{l,p}(\cdot)]^2} \tag{198}$$

$$\frac{\partial E_{k,ml,p}}{\partial \beta_{0,o}} = \frac{\exp_{l,p}(\cdot) \exp_{l,o}(\cdot)}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{l,p}(\cdot)]^2} - \frac{\exp_{m,p}(\cdot) \exp_{m,o}(\cdot)}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{m,p}(\cdot)]^2} \quad \forall\, o \neq p \tag{199}$$

$$\frac{\partial E_{k,ml,p}}{\partial \beta_{j,p}} = \frac{\exp_{m,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{m,o}(\cdot)) \cdot x_j}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{m,p}(\cdot)]^2} - \frac{\exp_{l,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{l,o}(\cdot)) \cdot x_j}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{l,p}(\cdot)]^2} \quad \forall\, j \in \{1,\ldots,K\} \setminus k, k+1 \tag{200}$$

$$\frac{\partial E_{k,ml,p}}{\partial \beta_{j,o}} = \frac{\exp_{l,p}(\cdot) \exp_{l,o}(\cdot) \cdot x_j}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{l,p}(\cdot)]^2} - \frac{\exp_{m,p}(\cdot) \exp_{m,o}(\cdot) \cdot x_j}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{m,p}(\cdot)]^2} \quad \forall\, o \neq p,\ j \in \{1,\ldots,K\} \setminus k, k+1 \tag{201}$$

$$\frac{\partial E_{k,ml,p}}{\partial \beta_{k,p}} = \frac{\exp_{m,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{m,o}(\cdot)) \cdot \bar{x}_k}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{m,p}(\cdot)]^2} - \frac{\exp_{l,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{l,o}(\cdot)) \cdot \bar{x}_k}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{l,p}(\cdot)]^2} \tag{202}$$

$$\frac{\partial E_{k,ml,p}}{\partial \beta_{k,o}} = \frac{\exp_{l,p}(\cdot) \exp_{l,o}(\cdot) \cdot \bar{x}_k}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{l,p}(\cdot)]^2} - \frac{\exp_{m,p}(\cdot) \exp_{m,o}(\cdot) \cdot \bar{x}_k}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{m,p}(\cdot)]^2} \tag{203}$$

$$\frac{\partial E_{k,ml,p}}{\partial \beta_{k+1,p}} = \frac{\exp_{m,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{m,o}(\cdot)) \cdot \overline{x_k^2}}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{m,p}(\cdot)]^2} - \frac{\exp_{l,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{l,o}(\cdot)) \cdot \overline{x_k^2}}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{l,p}(\cdot)]^2} \tag{204}$$

$$\frac{\partial E_{k,ml,p}}{\partial \beta_{k+1,o}} = \frac{\exp_{l,p}(\cdot) \exp_{l,o}(\cdot) \cdot \overline{x^2}_{km}}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{l,p}(\cdot)]^2} - \frac{\exp_{m,p}(\cdot) \exp_{m,o}(\cdot) \cdot \overline{x^2}_{km}}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{m,p}(\cdot)]^2} \tag{205}$$

where

$$\exp_{m,p}(\cdot) \equiv \exp\left(\beta_{0,p} + \sum_{j\in\{1,\ldots,K\}\setminus(k,k+1)} \beta_{j,p}x_j + \beta_{k,p}E[x_k|b_{m-1} < x_k \leq b_m] + \beta_{k+1,p}E[x_k^2|b_{m-1} < x_k \leq b_m]\right)$$

and

$$\exp_{l,p}(\cdot) \equiv \exp\left(\beta_{0,p} + \sum_{j\in\{1,\ldots,K\}\setminus(k,k+1)} \beta_{j,p}x_j + \beta_{k,p}E[x_k|b_{l-1} < x_k \leq b_l] + \beta_{k+1,p}E[x_k^2|b_{l-1} < x_k \leq b_l]\right).$$

In the case of the conditional logit the gradient vector, $\frac{\partial E_{k,ml,p}}{\partial \beta}$, modifies to:

$$\begin{aligned}\frac{\partial E_{t,ml,p}}{\partial \gamma_0} =& \frac{\exp_{m,p}(\cdot)\cdot\sum_{p\in C}\exp_{m,p}(\cdot) - \exp_{m,p}(\cdot)\cdot\sum_{p\in C}\exp_{m,p}(\cdot)}{\left(\sum_{p\in C}\exp_{m,p}(\cdot)\right)^2} \\ &- \frac{\exp_{l,p}(\cdot)\cdot\sum_{p\in C}\exp_{l,p}(\cdot) - \exp_{l,p}(\cdot)\cdot\sum_{p\in C}\exp_{l,p}(\cdot)}{\left(\sum_{p\in C}\exp_{l,p}(\cdot)\right)^2}\end{aligned} \tag{206}$$

$$\begin{aligned}\frac{\partial E_{t,ml,p}}{\partial \gamma_j} =& \frac{\exp_{m,p}(\cdot)z_{j,p}\cdot\sum_{p\in C}\exp_{m,p}(\cdot) - \exp_{m,p}(\cdot)\cdot\sum_{p\in C}(\exp_{m,p}(\cdot)z_{j,p})}{\left(\sum_{p\in C}\exp_{m,p}(\cdot)\right)^2} \\ &- \frac{\exp_{l,p}(\cdot)z_{j,p}\cdot\sum_{p\in C}\exp_{l,p}(\cdot) - \exp_{l,p}(\cdot)\cdot\sum_{p\in C}(\exp_{l,p}(\cdot)z_{j,p})}{\left(\sum_{p\in C}\exp_{l,p}(\cdot)\right)^2} \quad \forall\ \gamma_j\ \ j \neq t, t+1\end{aligned} \tag{207}$$

$$\begin{aligned}\frac{\partial E_{t,ml,p}}{\partial \gamma_t} =& \frac{\exp_{m,p}(\cdot)z_{t,p}\cdot\sum_{p\in C}\exp_{m,p}(\cdot) - \exp_{m,p}(\cdot)\cdot\sum_{p\in C}(\exp_{m,p}(\cdot)z_{t,p})}{\left(\sum_{p\in C}\exp_{m,p}(\cdot)\right)^2} \\ &- \frac{\exp_{l,p}(\cdot)z_{t,p}\cdot\sum_{p\in C}\exp_{l,p}(\cdot) - \exp_{l,p}(\cdot)\cdot\sum_{p\in C}(\exp_{l,p}(\cdot)z_{t,p})}{\left(\sum_{p\in C}\exp_{l,p}(\cdot)\right)^2}\end{aligned} \tag{208}$$

$$\begin{aligned}\frac{\partial E_{t,ml,p}}{\partial \gamma_{t+1}} =& \frac{\exp_{m,p}(\cdot)z_{t+1,p}\cdot\sum_{p\in C}\exp_{m,p}(\cdot) - \exp_{m,p}(\cdot)\cdot\sum_{p\in C}(\exp_{m,p}(\cdot)z_{t+1,p})}{\left(\sum_{p\in C}\exp_{m,p}(\cdot)\right)^2} \\ &- \frac{\exp_{l,p}(\cdot)z_{t+1,p}\cdot\sum_{p\in C}\exp_{l,p}(\cdot) - \exp_{l,p}(\cdot)\cdot\sum_{p\in C}(\exp_{l,p}(\cdot)z_{t+1,p})}{\left(\sum_{p\in C}\exp_{l,p}(\cdot)\right)^2}\end{aligned} \tag{209}$$

where

$$\exp_{m,p}(\cdot) \equiv \exp\left(\gamma_0 + \sum_{j\in\{1,\ldots,T\}\setminus(t,t+1)} \gamma_j z_{j,p} + \gamma_{t,p}E[z_t|b_{m-1} < z_t \leq b_m] + \gamma_{t+1,p}E[z_t^2|b_{m-1} < z_t \leq b_m]\right)$$

and

$$\exp_{l,p}(\cdot) \equiv \exp\left(\gamma_0 + \sum_{j\in\{1,\ldots,T\}\setminus(t,t+1)} \gamma_j z_{j,p} + \gamma_t E[z_t|b_{l-1} < z_t \leq b_l] + \gamma_t + 1E[z_t^2|b_{l-1} < z_t \leq b_l]\right).$$

Finally, in the case of the nested logit, the gradient vector, $\frac{\partial E_{k,ml,p}}{\partial \beta}$, modifies to:

$$\begin{aligned}
\frac{\partial E_{t,ml,p}}{\partial \beta_0} = & \left(\left(\frac{1}{\lambda_o} - \frac{\sum_{p \in B_o} \left(\exp_{m,p,o}(\cdot) \frac{1}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{m,p,o}(\cdot)} \right) \right. \\
& \left. + \left(\sum_{p \in B_o} 1 - \frac{\sum_{o=1}^{O} \left(\exp_{m,o}(\cdot) \sum_{p \in B_o} 1 \right)}{\sum_{o=1}^{O} \exp_{m,o}(\cdot)} \right) \right) \pi_{m,p} \pi_{m,o} \\
& - \left(\left(\frac{1}{\lambda_o} - \frac{\sum_{p \in B_o} \left(\exp_{l,p,o}(\cdot) \frac{1}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{l,p,o}(\cdot)} \right) \right. \\
& \left. + \left(\sum_{p \in B_o} 1 - \frac{\sum_{o=1}^{O} \left(\exp_{l,o}(\cdot) \sum_{p \in B_o} 1 \right)}{\sum_{o=1}^{O} \exp_{l,o}(\cdot)} \right) \right) \pi_{l,p} \pi_{l,o}
\end{aligned} \tag{210}$$

$$\begin{aligned}
\frac{\partial E_{t,ml,p}}{\partial \beta_j} = & \left(\left(\frac{x_{p,j}}{\lambda_o} - \frac{\sum_{p \in B_o} \left(\exp_{m,p,o}(\cdot) \frac{x_{p,j}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{m,p,o}(\cdot)} \right) \right. \\
& \left. + \left(\sum_{p \in B_o} x_{p,j} - \frac{\sum_{o=1}^{O} \left(\exp_{m,o}(\cdot) \sum_{p \in B_o} x_{p,j} \right)}{\sum_{o=1}^{O} \exp_{m,o}(\cdot)} \right) \right) \pi_{m,p} \pi_{m,o} \\
& - \left(\left(\frac{x_{p,j}}{\lambda_o} - \frac{\sum_{p \in B_o} \left(\exp_{l,p,o}(\cdot) \frac{x_{p,j}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{l,p,o}(\cdot)} \right) \right. \\
& \left. + \left(\sum_{p \in B_o} x_{p,j} - \frac{\sum_{o=1}^{O} \left(\exp_{l,o}(\cdot) \sum_{p \in B_o} x_{p,j} \right)}{\sum_{o=1}^{O} \exp_{l,o}(\cdot)} \right) \right) \pi_{l,p} \pi_{l,o}
\end{aligned} \tag{211}$$

$$\begin{aligned}
\frac{\partial E_{t,ml,p}}{\partial \beta_k} = & \left(\left(\frac{x_{p,m}}{\lambda_o} - \frac{\sum_{p \in B_o} \left(\exp_{m,p,o}(\cdot) \frac{x_{p,m}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{m,p,o}(\cdot)} \right) \right. \\
& \left. + \left(\sum_{p \in B_o} x_{p,m} - \frac{\sum_{o=1}^{O} \left(\exp_{m,o}(\cdot) \sum_{p \in B_o} x_{p,m} \right)}{\sum_{o=1}^{O} \exp_{m,o}(\cdot)} \right) \right) \pi_{m,p} \pi_{m,o} \\
& - \left(\left(\frac{x_{p,l}}{\lambda_o} - \frac{\sum_{p \in B_o} \left(\exp_{l,p,o}(\cdot) \frac{x_{p,l}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{l,p,o}(\cdot)} \right) \right. \\
& \left. + \left(\sum_{p \in B_o} x_{p,l} - \frac{\sum_{o=1}^{O} \left(\exp_{l,o}(\cdot) \sum_{p \in B_o} x_{p,l} \right)}{\sum_{o=1}^{O} \exp_{l,o}(\cdot)} \right) \right) \pi_{l,p} \pi_{l,o}
\end{aligned} \tag{212}$$

$$\frac{\partial E_{t,ml,p}}{\partial \beta_{k+1}} = \left(\left(\frac{x_{p,m+1}}{\lambda_o} - \frac{\sum_{p \in B_o} \left(\exp_{m,p,o}(\cdot) \frac{x_{p,m+1}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{m,p,o}(\cdot)} \right) + \left(\sum_{p \in B_o} x_{p,m+1} - \frac{\sum_{o=1}^{O} \left(\exp_{m,o}(\cdot) \sum_{p \in B_o} x_{p,m+1} \right)}{\sum_{o=1}^{O} \exp_{m,o}(\cdot)} \right) \right) \pi_{m,p} \pi_{m,o} - \left(\left(\frac{x_{p,l+1}}{\lambda_o} - \frac{\sum_{p \in B_o} \left(\exp_{l,p,o}(\cdot) \frac{x_{p,l+1}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{l,p,o}(\cdot)} \right) + \left(\sum_{p \in B_o} x_{p,l+1} - \frac{\sum_{o=1}^{O} \left(\exp_{l,o}(\cdot) \sum_{p \in B_o} x_{p,l+1} \right)}{\sum_{o=1}^{O} \exp_{l,o}(\cdot)} \right) \right) \pi_{l,p} \pi_{l,o} \tag{213}$$

with

$$\exp_{m,p,o}(\cdot) \equiv \exp \left(\beta_0 / \lambda_o + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j / \lambda_o x_{j,p} + \beta_k / \lambda_o E[x_{k,p} | b_{m-1} < x_{k,p} \leq b_m] \right) \tag{214}$$

$$\exp_{l,p,o}(\cdot) \equiv \exp \left(\beta_0 / \lambda_o + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j / \lambda_o x_{j,p} + \beta_k / \lambda_o E[x_{k,p} | b_{m-1} < x_{k,p} \leq b_m] \right) \tag{215}$$

$$\exp_{m/l,o}(\cdot) \equiv \exp \left(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o \right) \tag{216}$$

Using helper functions `EXSquared` (equation 41), `elaIntBounds` (checking arguments `refBound` and `intBound`), `checkXPos` (checking argument `xPos`, see appendix A), and `checkXBeta` (checking $\beta_0 + \sum_{j=1}^{K} \beta_j x_j$ for plausible values, see appendix A), the following function calculates the effect and its standard error according to equations (185), (187), and (194) to (197) for the binary logit, according to equations (188) and (198) to (205) for the multi-nomial logit, and according to equations (189) and (206) to (209) for the conditional logit:

```
logEffInt <- function( allCoef, allCoefBra = NA, allXVal, allXValBra=NA,
                       xPos, refBound, intBound, yCat, yCatBra, lambda,
                       allCoefSE = rep( NA, length( allCoef ) ),
                       method = "binary" ){
if( method == "binary" ){
  # number of coefficients
  nCoef <- length( allCoef )
  # check arguments
  if( length( allXVal ) != nCoef ){
    stop( "argument 'allCoef' and 'allXVal' must have the same length" )
  }
  if( length( allCoefSE ) != nCoef ){
```

```
    stop( "argument 'allCoef' and 'allCoefSE' must have the same length" )
  }
} else if( method == "MNL" ){
  # number of coefficients
    NCoef <- length( allCoef )
    mCoef <- matrix( allCoef, nrow = length( allXVal ))
    nCoef <- dim( mCoef )[1]
    pCoef <- dim( mCoef )[2]
   # check arguments
    if( length( allXVal ) != nCoef ){
      stop( "argument 'allCoef' and 'allXVal' must have the same length" )
    }
    if( length( allCoefSE ) != NCoef ){
      stop( "argument 'allCoef' and 'allCoefSE' must have the same length" )
    }
} else if( method == "CondL"){
  # number of coefficients
    nCoef <- length( allCoef )
    mXVal <- matrix( allXVal, nrow = nCoef )
    pCoef <- dim( mXVal )[2]
  # check arguments
    if( dim( mXVal )[1] != nCoef ){
      stop( "argument 'allCoef' and 'allXVal' must have the same length" )
    }
    if( length( allCoefSE ) != nCoef ){
      stop( "argument 'allCoef' and 'allCoefSE' must have the same length" )
    }
} else{
   nCoef <- length( allCoef )
   NCoef <- length( allCoefBra )
   mXValBra <- matrix( allXValBra, nrow = NCoef )
   nXValBra <- dim( mXValBra )[1]
   pXValBra <- dim( mXValBra )[2]
 # check arguments
   if( NCoef != nXValBra ){
     stop( "arguments 'allCoefBra' and 'allXValBra' must have the same length")
   }
   0 <- length( allXVal )
   nXVal <- unlist( lapply( allXVal, function(x) dim( x )[1] ) )
   pCoef <- unlist( lapply( allXVal, function(x) dim( x )[2] ) )
   if( nCoef != nXVal[ yCatBra ] ){
     stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
   }
   if( nCoef != length( allCoefSE) ){
     stop( "arguments 'allCoef' and 'allCoefSE' must have the same length" )
   }
```

```
}
  checkXPos( xPos, minLength = 1, maxLength = 2, minVal = 1, maxVal = nCoef )
  refBound <- elaIntBounds( refBound, 1, argName = "refBound" )
  intBound <- elaIntBounds( intBound, 1, argName = "intBound" )
if( method == "binary" || method == "MNL" ){
  if( any( !is.na( allXVal[ xPos ] ) ) ) {
    allXVal[ xPos ] <- NA
    warning( "values of argument 'allXVal[ xPos ]' are ignored",
      " (set these values to 'NA' to avoid this warning)" )
  }
}else if( method == "CondL" ){
  for( p in 1:pCoef ){
    if( any( !is.na( mXVal[ xPos, p ] ) ) ){
      mXVal[ xPos, p ] <- NA
      warning( "values of argument 'allXVal[ xPos ]' are ignored",
      " (set these values to 'NA' to avoid this warning)" )
    }
  }
}else{
  for( p in 1:pCoef[ yCatBra ] ){
    if( any( !is.na( allXVal[[ yCatBra ]][ xPos, p ] ) ) ){
      mXVal[ xPos, p ] <- NA
      warning( "values of argument 'allXVal[ xPos ]' are ignored",
      " (set these values to 'NA' to avoid this warning)" )
    }
  }
}
  # calculate xBars
  intX <- mean( intBound )
  refX <- mean( refBound )
  if( length( xPos ) == 2 ) {
    intX <- c( intX, EXSquared( intBound[1], intBound[2] ) )
    refX <- c( refX, EXSquared( refBound[1], refBound[2] ) )
  }
  if( length( intX ) != length( xPos ) || length( refX ) != length( xPos ) ) {
    stop( "internal error: 'intX' or 'refX' does not have the expected length" )
  }
  # define X' * beta
  if( method == "binary" ){
    intXbeta <- sum( allCoef * replace( allXVal, xPos, intX ) )
    refXbeta <- sum( allCoef * replace( allXVal, xPos, refX ) )
    checkXBeta( c( intXbeta, refXbeta ) )
  } else if( method == "MNL" ){
    intXbeta <- colSums( mCoef * replace( allXVal, xPos, intX ) )
    refXbeta <- colSums( mCoef * replace( allXVal, xPos, refX ) )
  } else if( method == "CondL" ){
```

```
    mXValint <- mXValref <- mXVal
    for( p in 1:pCoef ){
      mXValint[ ,p] <- replace( mXValint[ ,p], xPos, intX )
      mXValref[ ,p] <- replace( mXValref[ ,p], xPos, refX )
    }
    intXbeta <- colSums( allCoef * mXValint )
    refXbeta <- colSums( allCoef * mXValref )
  } else{
    mCoef <- matrix( rep( allCoef, 0 ), nrow = nCoef, 0 ) %*% diag( 1/ lambda )
    mXValint <- mXValref <- allXVal
    for( i in 1:0 ){
      for( p in 1:pCoef[i] ){
        mXValint[[i]][ ,p] <- replace( mXValint[[i]][ ,p], xPos, intX )
        mXValref[[i]][ ,p] <- replace( mXValref[[i]][ ,p], xPos, refX )
      }
    }
    refXbeta <- intXbeta <- rep( list( NA ), 0 )
    for( l in 1:0 ){
      intXbeta[[ l ]] <- colSums( mCoef[ ,l ] * mXValint[[ l ]] )
      refXbeta[[ l ]] <- colSums( mCoef[ ,l ] * mXValref[[ l ]] )
    }
    XbetaBra <- colSums( allCoefBra * mXValBra )
  }
  # effect E_{k,ml}
  if( method == "binary" ){
    eff <- exp( intXbeta )/( 1 + exp( intXbeta ) ) -
           exp( refXbeta )/( 1 + exp( refXbeta ) )
  } else if( method == "MNL" ){
    eff <- exp( intXbeta[ yCat ] )/( 1 + sum( exp( intXbeta ) ) ) -
           exp( refXbeta[ yCat ] )/( 1 + sum( exp( refXbeta ) ) )
  } else if( method == "CondL"){
    eff <- exp( intXbeta[ yCat ] )/( sum( exp( intXbeta ) ) ) -
           exp( refXbeta[ yCat ] )/( sum( exp( refXbeta ) ) )
  } else{
    intBranch <- refBranch <- rep( list( NA ), 0 )
    for( l in 1:0 ){
      intBranch[[ l ]] <- exp( XbetaBra[ l ] + lambda[ l ] *
                          log( sum( exp( intXbeta[[ l ]] ) ) ) )
      refBranch[[ l ]] <- exp( XbetaBra[ l ] + lambda[ l ] *
                          log( sum( exp( refXbeta[[ l ]] ) ) ) )
    }
    intBranch <- unlist( intBranch )
    refBranch <- unlist( refBranch )
    eff <- exp( intXbeta[[ yCatBra ]][ yCat ] )/( sum( exp( intXbeta[[ yCatBra ]] ) ) ) *
           intBranch[ yCatBra ]/ sum( intBranch ) -
           exp( refXbeta[[ yCatBra ]][ yCat ] )/( sum( exp( refXbeta[[ yCatBra ]] ) ) ) *
```

```
          refBranch[ yCatBra ]/ sum( refBranch )
}
# calculating approximate standard error
# partial derivative of E_{k,ml} w.r.t. all estimated coefficients
if( method == "binary" ){
  derivCoef <- rep( NA, nCoef )
  derivCoef[ -xPos ] <- ( exp( intXbeta )/( 1 + exp( intXbeta ) )^2 -
                          exp( refXbeta )/( 1 + exp( refXbeta ) )^2 ) *
                          allXVal[ -xPos ]
  derivCoef[ xPos ] <- exp( intXbeta )/( 1 + exp( intXbeta ) )^2 * intX -
                       exp( refXbeta )/( 1 + exp( refXbeta ) )^2 * refX
} else if( method == "MNL" ){
  derivCoef <- matrix( NA, nrow=nCoef, ncol=pCoef )
  for( p in 1:pCoef ){
    if( p == yCat ){
      derivCoef[ -xPos, p ] <-
        ( exp( intXbeta[ p ] ) *
        ( 1 + sum( exp( intXbeta[ -yCat ] ) ) )/
        ( 1 + sum( exp( intXbeta ) ) )^2 -
          exp( refXbeta[ p ] ) *
        ( 1 + sum( exp( refXbeta[ -yCat ] ) ) )/
        ( 1 + sum( exp( refXbeta ) ) )^2 ) * allXVal[ - xPos ]
      derivCoef[ xPos, p ] <-
        ( exp( intXbeta[ p ] ) *
        ( 1 + sum( exp( intXbeta[ -yCat ] ) ) )/
        ( 1 + sum( exp( intXbeta ) ) )^2 ) * intX -
        ( exp( refXbeta[ p ] ) *
        ( 1 + sum( exp( refXbeta[ -yCat ] ) ) )/
        ( 1 + sum( exp( refXbeta ) ) )^2 ) * refX
    } else{
      derivCoef[ -xPos, p ] <-
        ( ( exp( refXbeta[ yCat ] ) * exp( refXbeta[ p ] ) )/
          ( 1 + sum( exp( refXbeta ) ) )^2 -
          ( exp( intXbeta[ yCat ] ) * exp( intXbeta[ p ] ) )/
          ( 1 + sum( exp( intXbeta ) ) )^2 ) * allXVal[ -xPos ]
      derivCoef[ xPos, p ] <-
          ( ( exp( refXbeta[ yCat ] ) * exp( refXbeta[ p ] ) )/
            ( 1 + sum( exp( refXbeta ) ) )^2 ) * intX -
          ( ( exp( intXbeta[ yCat ] ) * exp( intXbeta[ p ] ) )/
            ( 1 + sum( exp( intXbeta ) ) )^2 ) * refX
    }
  }
  derivCoef <- c( derivCoef )
} else if( method == "CondL" ){
  derivCoef <- rep( NA, nCoef )
  derivCoef[ -xPos ] <- ( exp( intXbeta[ yCat] ) * mXVal[ -xPos, yCat] *
```

```
                          sum( exp( intXbeta ) ) -
                          exp( intXbeta[ yCat] ) * rowSums( exp( intXbeta ) *
                          mXVal[ -xPos, ] ) )/
                        ( sum( exp( intXbeta ) ) )^2 -
                        ( exp( refXbeta[ yCat] ) * mXVal[ -xPos, yCat] *
                          sum( exp( refXbeta ) ) -
                          exp( refXbeta[ yCat] ) * rowSums( exp( refXbeta ) *
                          mXVal[ -xPos, ] ) )/
                        ( sum( exp( refXbeta ) ) )^2
  derivCoef[ xPos ] <-  ( exp( intXbeta[ yCat] ) * intX *
                          sum( exp( intXbeta ) ) -
                          exp( intXbeta[ yCat] ) * sum( exp( intXbeta ) * intX ) )/
                        ( sum( exp( intXbeta ) ) )^2 -
                        ( exp( refXbeta[ yCat] ) * refX *
                          sum( exp( refXbeta ) ) -
                          exp( refXbeta[ yCat] ) * sum( exp( refXbeta ) * refX ) )/
                        ( sum( exp( refXbeta ) ) )^2
} else{
  derivCoef <- rep( NA, nCoef )
  PImp <- exp( intXbeta[[ yCatBra ]][ yCat ])/( sum( exp( intXbeta[[ yCatBra ]] ) ) )
  PIlp <- exp( refXbeta[[ yCatBra ]][ yCat ])/( sum( exp( refXbeta[[ yCatBra ]] ) ) )
  PImo <- intBranch[ yCatBra ]/ sum( intBranch )
  PIlo <- refBranch[ yCatBra ]/ sum( refBranch )
  Om <- matrix(
        unlist( lapply( allXVal, function(x) rowSums( x[ -xPos, , drop = FALSE ] ) ) ),
        ncol = 0 )
  derivCoef[ -xPos ] <- ( ( allXVal[[ yCatBra ]][ -xPos, yCat ]/lambda[ yCatBra ] -
                        ( rowSums(
                          ( allXVal[[ yCatBra ]][ -xPos, ]/lambda[ yCatBra ] ) %*%
                            diag( exp( intXbeta[[ yCatBra ]] ) ) ) )/
                        ( sum( exp( intXbeta[[ yCatBra ]] ) ) ) ) +
                        ( rowSums( allXVal[[ yCatBra ]][ -xPos, ] ) -
                        ( rowSums( Om %*% diag( exp( intBranch ) ) )/
                        ( sum( intBranch ) ) ) ) * PImp * PImo -
                        ( ( allXVal[[ yCatBra ]][ -xPos, yCat ]/lambda[ yCatBra ] -
                        ( rowSums(
                          ( allXVal[[ yCatBra ]][ -xPos, ]/lambda[ yCatBra ] ) %*%
                            diag( exp( refXbeta[[ yCatBra ]] ) ) ) )/
                        ( sum( exp( refXbeta[[ yCatBra ]] ) ) ) ) +
                        ( rowSums( allXVal[[ yCatBra ]][ -xPos, ] ) -
                        ( rowSums( Om %*% diag( exp( refBranch ) ) )/
                        ( sum( refBranch ) ) ) ) * PIlp * PIlo
  derivCoef[ xPos ] <-  ( ( intX/lambda[ yCatBra ] -
                        ( sum( intX/lambda[ yCatBra ]  *
                               exp( intXbeta[[ yCatBra ]] ) ) )/
                        ( sum( exp( intXbeta[[ yCatBra ]] ) ) ) ) +
```

```
                        ( intX * pCoef[ yCatBra ] -
                        ( sum( intX * exp( intBranch ) )/
                        ( sum( intBranch ) ) ) ) * PImp * PImo -
                        ( ( refX/lambda[ yCatBra ] -
                        ( sum( refX/lambda[ yCatBra ]  *
                              exp( refXbeta[[ yCatBra ]] ) ) )/
                        ( sum( exp( refXbeta[[ yCatBra ]] ) ) ) ) +
                        ( refX * pCoef[ yCatBra ] -
                        ( sum( refX * exp( refBranch ) )/
                        ( sum( refBranch ) ) ) ) * PImp * PImo
  }
  # variance covariance of the coefficients (covariances set to zero)
  vcovCoef <- diag( allCoefSE^2 )
  # approximate standard error of the effect
  effSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  # object to be returned
  result <- c( effect = eff, stdEr = effSE )
  return( result )
}
```

where

- arguments `allCoef` $= (\beta_0, \ldots, \beta_K)^\top$ a vector of all coefficients from a binary logit model, `allCoef` $= (\gamma_0, \ldots, \gamma_T)^\top$ a vector of all coefficients from a conditional logit model, `allCoefBra` $= (\beta_0, \ldots, \beta_K)^\top$, a vector of all coefficients from a nested logit model, where `allCoefBra`, the coefficients at the branch level, must always be included in combination with `allCoef`, the coefficients at the twig level, or `allCoef` $= (\beta_{01}, \ldots, \beta_{K1}, \ldots, \beta_{0P}, \ldots, \beta_{KP})$ a vector of the P sets of coefficients from the multinomial logit regression which does **not** include any values for the reference category;

- argument `allXVal` $= (1, \ldots, \bar{x}_K)^\top$, a vector of all corresponding sample means and 1 for the intercept for the binary or multi-nomial logit, `allXVal` $= (1, \ldots, \bar{z}_{T,1}, \ldots, 1, \ldots, \bar{z}_{T,P})$ a vector of the P sets of explanatory variables from the conditional logit, or `allXVal` $= (((1, \ldots, \bar{x}_{K,p,o}), \ldots, (1, \ldots, \bar{x}_{K,P,o}))^\top, \ldots, ((1, \ldots, \bar{x}_{K,p,O}), \ldots, (1, \ldots, \bar{x}_{K,P,O}))^\top)^\top$, a list where the list elements are the corresponding matrices of the sample means for each nest at the twig level,[9] and which must always be combined with the corresponding values at the branch level in `allXVal`;

- argument `allCoefSE` $= (\mathrm{se}(\beta_0), \ldots, \mathrm{se}(\beta_K))^\top$, a vector of standard errors for all coefficients of a binary or conditional logit regression or of the standard errors of the P sets of coefficients in a multi-nomial logit regression;

- argument `xPos` $= k + 1$ or $(k + 1, k + 2)^\top$ a scalar or vector of two elements indicating the position of the coefficients of interest in vectors `allCoef` and `allXVal`;

- argument `refBound` $= (b_{l-1}, b_l)$ indicates the boundaries of the reference interval;

[9] The reason why we make an exception from the usual vector in the case of the nested logit model is that nests, o, can have different dimensions wrt P.

- argument `intBound` $= (b_{m-1}, b_m)$ indicates the boundaries of the interval of interest;
- argument `method = c("binary", "MNL", "CondL", "NestL" )` indicating the estimator used, where option `"CondL"` can also be used to calculate the semi-elasticity and standard error of a nested logit at the branch level and where option `"NestL"` estimates the semi-elasticity and standard error of the combined probabilities at the branch and twig level;
- argument `"lambda"` is a vector of the O parameter of the correlation coefficient between the the branch specific alternatives of interest, it is only relevant under option `"NestL"` and should be set to 1 in case the estimation coefficients in `allCoef` are already corrected by $\frac{\beta_k}{\lambda_o}$;
- finally, `yCat` a scalar identifying which of the P output categories is of interest, only in combination with `method = "MNL"`, `"CondL"`, and `"NestL"` (for the twig level), and `yCatBra` at the branch level.

```
# Example
eff6a <- logEffInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                    allXVal = c( 1, NA, 0.16, 0.13 ),
                    xPos = 2,
                    refBound = c( 8, 12 ), intBound = c( 13, 15 ) )
eff6a

##    effect     stdEr
## 0.0386895        NA

eff6b <- logEffInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                    allXVal = c( 1, NA, 0.16, 0.13 ),
                    xPos = 2,
                    refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                    allCoefSE = c( 0.003, 0.045, 0.007, 0.009 ) )
eff6b

##     effect      stdEr
## 0.03868950 0.01057185

# Example
eff7a <- logEffInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                    allXVal = c( 1, NA, NA, 0.0004 ),
                    xPos = c( 2, 3 ),
                    refBound = c( 8, 12 ), intBound = c( 13, 15 ))

## Warning in checkXBeta(c(intXbeta, refXbeta)):  At least one x'beta has an
implausible value:  13.2269066666667, 7.59690666666667

eff7a
```

```
##       effect        stdEr
## 0.0004999485           NA

eff7b <- logEffInt( allCoef = c( 0.33, 0.22, 0.05, 0.6 ),
                    allXVal = c( 1, NA, NA, 0.13 ),
                    xPos = c( 2, 3 ),
                    refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                    allCoefSE = c( 0.003, 0.045, 0.007, 0.009 ) )

## Warning in checkXBeta(c(intXbeta, refXbeta)):  At least one x'beta has an
implausible value:  13.3046666666667, 7.67466666666667

eff7b

##       effect        stdEr
## 0.0004625630 0.0003872931

#Example
eff8a <- logEffInt( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                    allXVal = c( 1, NA, 0.12 ),
                    xPos = 2,
                    refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                    yCat = 2, method = "MNL" )
eff8a

##      effect       stdEr
## -0.02698526          NA

eff8b <- logEffInt( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                    allXVal = c( 1, NA, 0.12 ),
                    xPos = 2,
                    refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                    yCat = 2,
                    allCoefSE = c( 0.003, 0.045, 0.007, 0.009, 0.0008, 0.9 ),
                    method = "MNL" )
eff8b

##      effect       stdEr
## -0.02698526  0.01805266

#Example
eff9a <- logEffInt( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                    allXVal = c( 1, NA, NA ),
                    xPos = c( 2, 3 ),
                    refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                    yCat = 2, method = "MNL" )
eff9a
```

```
##       effect        stdEr
## 0.0008810153           NA

eff9b <- logEffInt( allCoef = c( 0.2, 0.3, 0.5, -0.2, 0.03, 0.6 ),
                    allXVal = c( 1, NA, NA ),
                    xPos = c( 2, 3 ),
                    refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                    yCat = 2,
                    allCoefSE = c( 0.003, 0.045, 0.007, 0.009, 0.0008, 0.9 ),
                    method = "MNL" )
eff9b

##       effect        stdEr
## 0.0008810153 0.0802721893

#Example
eff10a <- logEffInt( allCoef = c( 0.2, 0.3, 0.5, 0.091 ),
                     allXVal = c( 1, NA, NA, 2.45, 1, NA, NA, 0.79 ),
                     xPos = c( 2, 3 ),
                     refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                     yCat = 2, method = "CondL" )
eff10a

##       effect        stdEr
## 5.551115e-17           NA

eff10b <- logEffInt( allCoef = c( 0.2, 0.3, 0.5, 0.091 ),
                     allXVal = c( 1, NA, NA, 2.45, 1, NA, NA, 0.79 ),
                     xPos = c( 2, 3 ),
                     refBound = c( 8, 12 ), intBound = c( 13, 15 ),
                     allCoefSE = c( 0.003, 0.045, 0.007, 0.009 ),
                     yCat = 2, method = "CondL" )
eff10b

##       effect        stdEr
## 5.551115e-17 8.894225e-01

# Example
matrix1 <- matrix( c( 1, NA, 0.3, 0.09, 1, NA, 0.9, 1.8 ), nrow = 4 )
matrix2 <- matrix( c( 1, NA, 0.099, 0.211 ), nrow = 4 )
eff12a <- logEffInt( allCoefBra = c( 0.445, 0.03, -0.002 ),
                     allCoef = c( 1.8, 0.005, -0.12, 0.8 ),
                     allXValBra = c( 1, 3.3, 4.5, 1, 0.1, 0.987 ),
                     allXVal = list( matrix1, matrix2 ),
                     refBound = c( 0.5, 1.5 ), intBound = c( 2, 3.5 ),
                     xPos = 2, yCatBra = 1, yCat = 2, lambda = c( 0.8, 1 ),
                     method = "NestedL" )
eff12a
```

```
##       effect        stdEr
## 1.110223e-16           NA

matrix1 <- matrix( c( 1, NA, 0.3, 0.09, 1, NA, 0.9, 1.8 ), nrow = 4 )
matrix2 <- matrix( c( 1, NA, 0.099, 0.211 ), nrow = 4 )
eff12b <- logEffInt( allCoefBra = c( 0.445, 0.03, -0.002 ),
                     allCoef = c( 1.8, 0.005, -0.12, 0.8 ),
                     allXValBra = c( 1, 3.3, 4.5, 1, 0.1, 0.987 ),
                     allXVal = list( matrix1, matrix2 ),
                     allCoefSE = c( 0.003, 0.045, 0.007, 0.0032 ),
                     refBound = c( 0.5, 1.5 ), intBound = c( 2, 3.5 ),
                     xPos = 2, yCatBra = 1, yCat = 2, lambda = c( 0.8, 1 ),
                     method = "NestedL" )
eff12b

##       effect        stdEr
## 1.110223e-16 8.904229e+16
```

3.4. Grouping and re-basing categorical variables

We consider a regression model

$$\Pr(y=1|x) = \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m\in\{1,\ldots,M\}\setminus m^*}\delta_m D_m\right)}{1+\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m\in\{1,\ldots,M\}\setminus m^*}\delta_m D_m\right)} \tag{217}$$

$$D_m = \begin{cases} 1 & \text{if } x_k \in c_m \\ 0 & \text{otherwise} \end{cases} \quad \forall\, m = 1,\ldots,M. \tag{218}$$

Like in section 1.4, the kth xplanatory variable is coded into M mutually exclusive categories $c_1,\ldots,c_M$, with $c_m \cap c_l = \emptyset\ \forall\ m \neq l$, and $D_1,\ldots,D_M$ the corresponding dummy variables.

In the case of a binary logit regression, equation (52) modifies to

$$E_{k,lr} = E[y|x_k \in c_l^*] - E[y|x_k \in c_r^*] \tag{219}$$

$$= \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m=1}^{M}\delta_m E[D_m|x_k \in c_l^*]\right)}{1+\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m=1}^{M}\delta_m E[D_m|x_k \in c_l^*]\right)} - \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m=1}^{M}\delta_m E[D_m|x_k \in c_r^*]\right)}{1+\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m=1}^{M}\delta_m E[D_m|x_k \in c_r^*]\right)} \tag{220}$$

$$= \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m=1}^{M}\delta_m D_{ml}\right)}{1+\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m=1}^{M}\delta_m D_{ml}\right)} - \frac{\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m=1}^{M}\delta_m D_{mr}\right)}{1+\exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k}\beta_j x_j + \sum_{m=1}^{M}\delta_m D_{mr}\right)} \tag{221}$$

where D_{mn} is defined as in equation (56). Whilst for the case of the multi-nomial logit regression, equation (52) modifies to

$$E_{k,lr,p} = \frac{\exp\left(\beta_{0,p} + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_{j,p} x_j + \sum_{m=1}^{M} \delta_{m,p} D_{ml}\right)}{1 + \sum_{p\in\{1,\dots,P\}\setminus p^*} \exp\left(\beta_{0,p} + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_{j,p} x_j + \sum_{m=1}^{M} \delta_{m,p} D_{ml}\right)} - \frac{\exp\left(\beta_{0,p} + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_{j,p} x_j + \sum_{m=1}^{M} \delta_{m,p} D_{mr}\right)}{1 + \sum_{p\in\{1,\dots,P\}\setminus p^*} \exp\left(\beta_{0,p} + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_{j,p} x_j + \sum_{m=1}^{M} \delta_{m,p} D_{mr}\right)} \tag{222}$$

and for the case of the conditional logit regression, equation (52) modifies to

$$E_{t,lr,p} = \frac{\exp\left(\gamma_0 + \sum_{j\in\{1,\dots,T\}\setminus t} \gamma_j z_{j,p} + \sum_{m=1}^{M} \delta_m D_{ml}\right)}{\sum_{p\in C} \exp\left(\gamma_0 + \sum_{j\in\{1,\dots,T\}\setminus t} \gamma_j z_{j,p} + \sum_{m=1}^{M} \delta_m D_{ml}\right)} - \frac{\exp\left(\gamma_0 + \sum_{j\in\{1,\dots,T\}\setminus t} \gamma_j z_{j,p} + \sum_{m=1}^{M} \delta_m D_{mr}\right)}{\sum_{p\in C} \exp\left(\gamma_0 + \sum_{j\in\{1,\dots,T\}\setminus t} \gamma_j z_{j,p} + \sum_{m=1}^{M} \delta_m D_{mr}\right)} \tag{223}$$

and for the case of the nested logit regression, equation (52) modifies to

$$\begin{aligned} E_{t,lr,p,o} = {} & \frac{\exp(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \sum_{m=1}^{M} \delta_m/\lambda_o D_{ml})}{\sum_{p\in B_o} \exp(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \sum_{m=1}^{M} \delta_m/\lambda_o D_{ml})} \\ & \cdot \frac{\exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_{o,l})}{\sum_{o=1}^{O} \exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_{o,l})} \\ & - \frac{\exp(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \sum_{m=1}^{M} \delta_m/\lambda_o D_{mr})}{\sum_{p\in B_o} \exp(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \sum_{m=1}^{M} \delta_m/\lambda_o D_{mr})} \\ & \cdot \frac{\exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_{o,r})}{\sum_{o=1}^{O} \exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_{o,r})} \end{aligned} \tag{224}$$

where

$$IV_{o,l/r} = \ln \sum_{p\in B_o} \exp\left(\beta_0/\lambda_o + \sum_{j\in\{1,\dots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \sum_{m=1}^{M} \gamma_m/\lambda_0 D_{m,l/r}\right). \tag{225}$$

In order to calculate the approximate standard error of the new effect $E_{k,lr}$, we again apply the Delta method:

$$\operatorname{se}(E_{k,lr}) = \sqrt{\left(\frac{\partial E_{k,lr}}{\partial \begin{pmatrix}\beta\\ \delta\end{pmatrix}}\right)^{\top} \cdot \operatorname{Var}\begin{pmatrix}\beta\\ \delta\end{pmatrix} \cdot \frac{\partial E_{k,lr}}{\partial \begin{pmatrix}\beta\\ \delta\end{pmatrix}}}, \tag{226}$$

with the partial derivatives defined as

$$\frac{\partial E_{k,lr}}{\partial \beta_0} = \frac{\exp_{ml}(\cdot)}{[1+\exp_{ml}(\cdot)]^2} - \frac{\exp_{mr}(\cdot)}{[1+\exp_{mr}(\cdot)]^2} \tag{227}$$

$$\frac{\partial E_{k,lr}}{\partial \beta_j} = \left(\frac{\exp_{ml}(\cdot)}{[1+\exp_{ml}(\cdot)]^2} - \frac{\exp_{mr}(\cdot)}{[1+\exp_{mr}(\cdot)]^2} \right) \cdot \bar{x}_j \; \forall \; j \in \{1, \ldots, K\} \setminus k \tag{228}$$

$$\frac{\partial E_{k,lr}}{\partial \delta_m} = \frac{\exp_{ml}(\cdot)}{[1+\exp_{ml}(\cdot)]^2} D_{ml} - \frac{\exp_{mr}(\cdot)}{[1+\exp_{mr}(\cdot)]^2} D_{mr} \; \forall \; m = 1, \ldots, M. \tag{229}$$

and for the multi-nomial logit as

$$\begin{aligned} \frac{\partial E_{k,lr,p}}{\partial \beta_{0,p}} =& \frac{\exp_{ml,p}(\cdot) \cdot (1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{ml,o}(\cdot))}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{ml,p}(\cdot)]^2} \\ & - \frac{\exp_{mr,p}(\cdot) \cdot (1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{mr,o}(\cdot))}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{mr,p}(\cdot)]^2} \end{aligned} \tag{230}$$

$$\begin{aligned} \frac{\partial E_{k,lr,p}}{\partial \beta_{0,o}} =& \frac{\exp_{mr,p}(\cdot) \exp_{mr,o}(\cdot)}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{mr,p}(\cdot)]^2} \\ & - \frac{\exp_{ml,p}(\cdot) \exp_{ml,o}(\cdot)}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{ml,p}(\cdot)]^2} \quad \forall \; o \neq p \end{aligned} \tag{231}$$

$$\begin{aligned} \frac{\partial E_{k,lr,p}}{\partial \beta_{j,p}} =& \frac{\exp_{ml,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{ml,o}(\cdot)) \cdot x_j}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{ml,p}(\cdot)]^2} \\ & - \frac{\exp_{mr,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{mr,o}(\cdot)) \cdot x_j}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{mr,p}(\cdot)]^2} \quad \forall \; j \in \{1, \ldots, K\} \setminus k, k+1 \end{aligned} \tag{232}$$

$$\begin{aligned} \frac{\partial E_{k,lr,p}}{\partial \beta_{j,o}} =& \frac{\exp_{mr,p}(\cdot) \exp_{mr,o}(\cdot) \cdot x_j}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{mr,p}(\cdot)]^2} \\ & - \frac{\exp_{ml,p}(\cdot) \exp_{ml,o}(\cdot) \cdot x_j}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{ml,p}(\cdot)]^2} \quad \forall \; o \neq p, \; j \in \{1, \ldots, K\} \setminus k, k+1 \end{aligned} \tag{233}$$

$$\begin{aligned} \frac{\partial E_{k,lr,p}}{\partial \delta_{m,p}} =& \frac{\exp_{ml,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{ml,o}(\cdot)) \cdot D_{ml}}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{ml,p}(\cdot)]^2} \\ & - \frac{\exp_{mr,p}(\cdot)(1 + \sum_{o \in \{1,\ldots,P\} \setminus p,p^*} \exp_{mr,o}(\cdot)) \cdot D_{mr}}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{mr,p}(\cdot)]^2} \end{aligned} \tag{234}$$

$$\frac{\partial E_{k,ml,p}}{\partial \delta_{m,o}} = \frac{\exp_{mr,p}(\cdot) \exp_{mr,o}(\cdot) \cdot D_{mr}}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{mr,p}(\cdot)]^2} - \frac{\exp_{ml,p}(\cdot) \exp_{ml,o}(\cdot) \cdot D_{ml}}{[1 + \sum_{p \in \{1,\ldots,P\} \setminus p^*} \exp_{ml,p}(\cdot)]^2} \tag{235}$$

with

$$\exp_{ml}(\cdot) \equiv \exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{ml}\right) \tag{236}$$

$$\exp_{mr}(\cdot) \equiv \exp\left(\beta_0 + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{mr}\right) \tag{237}$$

and for the conditional logit as

$$\frac{\partial E_{t,lr}}{\partial \gamma_0} = \frac{\exp_{ml,p}(\cdot)\cdot\sum_{p\in C}\exp_{ml,p} - \exp_{ml,p}(\cdot)\cdot\sum_{p\in C}\exp_{ml,p}}{[\sum_{p\in C}\exp_{ml}(\cdot)]^2} - \frac{\exp_{mr,p}(\cdot)\cdot\sum_{p\in C}\exp_{mr,p} - \exp_{mr,p}(\cdot)\cdot\sum_{p\in C}\exp_{mr,p}}{[\sum_{p\in C}\exp_{mr}(\cdot)]^2} \tag{238}$$

$$\frac{\partial E_{t,lr}}{\partial \gamma_j} = \frac{\exp_{ml,p}(\cdot) z_{j,p}\cdot\sum_{p\in C}\exp_{ml,p} - \exp_{ml,p}(\cdot)\cdot\sum_{p\in C}\exp_{ml,p}\, z_{j,p}}{[\sum_{p\in C}\exp_{ml}(\cdot)]^2} - \frac{\exp_{mr,p}(\cdot) z_{j,p}\cdot\sum_{p\in C}\exp_{mr,p} - \exp_{mr,p}(\cdot)\cdot\sum_{p\in C}\exp_{mr,p}\, z_{j,p}}{[\sum_{p\in C}\exp_{mr}(\cdot)]^2} \;\forall\, j \in \{1,\ldots,T\}\setminus t \tag{239}$$

$$\frac{\partial E_{t,lr}}{\partial \delta_m} = \frac{\exp_{ml,p}(\cdot) D_{ml}\cdot\sum_{p\in C}\exp_{ml,p} - \exp_{ml,p}(\cdot)\cdot\sum_{p\in C}\exp_{ml,p}\, D_{ml}}{[\sum_{p\in C}\exp_{ml}(\cdot)]^2} - \frac{\exp_{mr,p}(\cdot) D_{mr}\cdot\sum_{p\in C}\exp_{mr,p} - \exp_{mr,p}(\cdot)\cdot\sum_{p\in C}\exp_{mr,p}\, D_{mr}}{[\sum_{p\in C}\exp_{mr}(\cdot)]^2} \;\forall\, m = 1,\ldots,M \tag{240}$$

(241)

and for the nested logit as

$$\frac{\partial E_{o,p,lr}}{\partial \beta_0} = \left(\left(\frac{1}{\lambda_o} - \frac{\sum_{p \in B_o} (\exp_{o,p,l}(\cdot) \frac{1}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,l}(\cdot)} \right) + \left(\frac{\sum_{p \in B_o} (\exp_{o,p,l}(\cdot) \frac{1}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,l}(\cdot)} - \frac{\sum_{o=1}^{O} \left(\exp_{o,l}(\cdot) \frac{\sum_{p \in B_o} (\exp o,p,l(\cdot) 1/\lambda_o)}{\sum_{p \in B_o \exp_{o,p,l}(\cdot)}} \right)}{\sum_{o=1}^{O} \exp_{o,l}(\cdot)} \right) \right) \cdot \pi_{p,l} \pi_{o,l} - \left(\left(\frac{1}{\lambda_o} - \frac{\sum_{p \in B_o} (\exp_{o,p,r}(\cdot) \frac{1}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,r}(\cdot)} \right) + \left(\frac{\sum_{p \in B_o} (\exp_{o,p,r}(\cdot) \frac{1}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,r}(\cdot)} - \frac{\sum_{o=1}^{O} \left(\exp_{o,r}(\cdot) \frac{\sum_{p \in B_o} (\exp o,p,r(\cdot) 1/\lambda_o)}{\sum_{p \in B_o \exp_{o,p,r}(\cdot)}} \right)}{\sum_{o=1}^{O} \exp_{o,r}(\cdot)} \right) \right) \cdot \pi_{p,r} \pi_{o,r} \tag{242}$$

$$\frac{\partial E_{o,p,lr}}{\partial \beta_j} = \left(\left(\frac{x_{j,p}}{\lambda_o} - \frac{\sum_{p \in B_o} (\exp_{o,p,l}(\cdot) \frac{x_{j,p}}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,l}(\cdot)} \right) + \left(\frac{\sum_{p \in B_o} (\exp_{o,p,l}(\cdot) \frac{x_{p,j}}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,l}(\cdot)} - \frac{\sum_{o=1}^{O} \left(\exp_{o,l}(\cdot) \frac{\sum_{p \in B_o} (\exp o,p,l(\cdot) x_{j,p}/\lambda_o)}{\sum_{p \in B_o \exp_{o,p,l}(\cdot)}} \right)}{\sum_{o=1}^{O} \exp_{o,l}(\cdot)} \right) \right) \cdot \pi_{p,l} \pi_{o,l} - \left(\left(\frac{x_{j,p}}{\lambda_o} - \frac{\sum_{p \in B_o} (\exp_{o,p,r}(\cdot) \frac{x_{j,p}}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,r}(\cdot)} \right) + \left(\frac{\sum_{p \in B_o} (\exp_{o,p,r}(\cdot) \frac{x_{p,j}}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,r}(\cdot)} - \frac{\sum_{o=1}^{O} \left(\exp_{o,r}(\cdot) \frac{\sum_{p \in B_o} (\exp o,p,r(\cdot) x_{j,p}/\lambda_o)}{\sum_{p \in B_o \exp_{o,p,r}(\cdot)}} \right)}{\sum_{o=1}^{O} \exp_{o,r}(\cdot)} \right) \right) \cdot \pi_{p,r} \pi_{o,r} \tag{243}$$

$$\frac{\partial E_{o,p,lr}}{\partial \delta_m} = \left(\left(\frac{D_{m,p,l}}{\lambda_o} - \frac{\sum_{p \in B_o} (\exp_{o,p,l}(\cdot) \frac{D_{m,p,l}}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,l}(\cdot)} \right) + \left(\frac{\sum_{p \in B_o} (\exp_{o,p,l}(\cdot) \frac{D_{m,p,l}}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,l}(\cdot)} - \frac{\sum_{o=1}^{O} \left(\exp_{o,l}(\cdot) \frac{\sum_{p \in B_o} (\exp o,p,l(\cdot) D_{m,p,l}/\lambda_o)}{\sum_{p \in B_o \exp_{o,p,l}(\cdot)}} \right)}{\sum_{o=1}^{O} \exp_{o,l}(\cdot)} \right) \right) \cdot \pi_{p,l} \pi_{o,l} - \left(\left(\frac{x_{j,p}}{\lambda_o} - \frac{\sum_{p \in B_o} (\exp_{o,p,r}(\cdot) \frac{D_{m,p,r}}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,r}(\cdot)} \right) + \left(\frac{\sum_{p \in B_o} (\exp_{o,p,r}(\cdot) \frac{D_{m,p,r}}{\lambda_o})}{\sum_{p \in B_o} \exp_{o,p,r}(\cdot)} - \frac{\sum_{o=1}^{O} \left(\exp_{o,r}(\cdot) \frac{\sum_{p \in B_o} (\exp o,p,r(\cdot) D_{m,p,r}/\lambda_o)}{\sum_{p \in B_o \exp_{o,p,r}(\cdot)}} \right)}{\sum_{o=1}^{O} \exp_{o,r}(\cdot)} \right) \right) \cdot \pi_{p,r} \pi_{o,r} \tag{244}$$

with

$$\exp_{p,o,l}(\cdot) \equiv \exp\left(\beta_0/\lambda_o + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \sum_{m=1}^{M} \delta_m/\lambda_o D_{m,p,l}\right) \tag{245}$$

$$\exp_{p,o,r}(\cdot) \equiv \exp\left(\beta_0/\lambda_o + \sum_{j\in\{1,\ldots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \sum_{m=1}^{M} \delta_m/\lambda_o D_{m,p,l}\right) \tag{246}$$

$$\exp_{l/r,o}(\cdot) \equiv \exp\left(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o\right) \tag{247}$$

The following function calculates the effect $E_{k,lr}$ and its standard error in accordance with equations (221), and (226) to (237) for the binary logit, effect $E_{k,lr,p}$ and its standard error in accordance with equations (222) and (230) to (235) for the multi-nomial logit, and effect $E_{t,lr,p}$ and its standard error in accordance with equations (223) and (238) to (241):

```
logEffGr <- function( allCoef, allXVal, xPos, Group, yCat = NA,
                      allCoefSE = rep( NA, length( allCoef ) ),
                      method = "binary" ){
  if( method == "binary" ){
     nCoef <- length( allCoef )
     xCoef <- allCoef[ xPos ]
     xShares <- allXVal[ xPos ]
  } else if( method == "MNL" ){
     nCoef <- length( allCoef )
     mCoef <- matrix( allCoef, nrow = length( allXVal ) )
     NCoef <- dim( mCoef )[2]
     pCoef <- dim( mCoef )[1]
     xCoef <- mCoef[ xPos, ]
     xShares <- allXVal[ xPos ]
  } else{
     nCoef <- length( allCoef )
     xCoef <- allCoef[ xPos ]
     mXVal <- matrix( allXVal, nrow = nCoef )
     pCoef <- dim( mXVal )[2]
     xShares <- mXVal[ xPos, ]
  }
  if( method == "binary" || method == "MNL" ){
    if( sum( xShares ) > 1 ){
      stop( "the shares in argument 'xShares' sum up to a value larger than 1" )
    }
  } else{
    for( p in 1:pCoef ){
      if( sum( xShares[ , p ] ) > 1 ){
        stop( "the shares in argument 'xShares' sum up to a value larger than 1" )
```

```
    }
  }
}
if( method == "binary" ){
  if( length( xCoef ) != length( xShares ) ){
    stop( "arguments 'xCoef' and 'xShares' must have the same length" )
  }
  if( length( xCoef ) != length( Group ) ){
    stop( "arguments 'xCoef' and 'Group' must have the same length" )
  }
} else if( method == "MNL" ){
  if( dim( xCoef )[1] != length( xShares ) ){
    stop( "arguments 'xCoef' and 'xShares' must have the same length" )
  }
  if( dim( xCoef )[1] != length( Group ) ){
    stop( "arguments 'xCoef' and 'Group' must have the same length" )
  }
} else{
  if( length( xCoef ) != dim( xShares )[1] ){
    stop( "arguments 'xCoef' and 'xShares' must have the same length" )
  }
  if( length( xCoef ) != length( Group ) ){
    stop( "arguments 'xCoef' and 'Group' must have the same length" )
  }
}
if( !all( Group %in% c( -1, 0, 1 ) ) ){
  stop( "all elements of argument 'Group' must be -1, 0, or 1" )
}
if( method == "binary" ){
  # D_mr
  DRef <- sum( xCoef[ Group == -1 ] * xShares[ Group == -1 ]) /
          sum( xShares[ Group == -1 ] )
  XBetaRef <- sum( allCoef[ -xPos ] * allXVal[ -xPos ]) + DRef
  # D_ml
  DEffect <- sum( xCoef[ Group == 1 ] * xShares[ Group == 1 ]) /
             sum( xShares[ Group == 1 ] )
  XBetaEffect <- sum( allCoef[ -xPos ] * allXVal[ -xPos ]) + DEffect
  # effect
  effeG <- exp( XBetaEffect )/( 1 + exp( XBetaEffect ) ) -
           exp( XBetaRef )/( 1 + exp( XBetaEffect ) )
} else if( method == "MNL" ){
  # D_mr
  DRef <- colSums( xCoef[ Group == -1, , drop = FALSE ] *
                   xShares[ Group == -1 ] )/
              sum( xShares[ Group == -1 ] )
  XBetaRef <- colSums( mCoef[ -xPos, , drop = FALSE ] *
```

```
                        allXVal[ -xPos ]) + DRef
  # D_ml
  DEffect <- colSums( xCoef[ Group == 1, , drop = FALSE ] *
                      xShares[ Group == 1 ] )/
                  sum( xShares[ Group == 1 ] )
  XBetaEffect <- colSums( mCoef[ -xPos, , drop = FALSE ] *
                          allXVal[ -xPos ]) + DEffect
  # effect
  effeG <- exp( XBetaEffect[ yCat ] )/( 1 + sum( exp( XBetaEffect ) ) ) -
           exp( XBetaRef[ yCat ] )/( 1 + sum( exp( XBetaRef ) ) )
} else{
  # D_mr
  DRef <- colSums( xCoef[ Group == -1 ] *
                   xShares[ Group == -1, , drop = FALSE ] )/
              sum( xShares[ Group == -1, , drop = FALSE ] )
  XBetaRef <- colSums( allCoef[ -xPos ] *
                       mXVal[ -xPos, , drop = FALSE ] ) + DRef
  # D_ml
  DEffect <- colSums( xCoef[ Group == 1 ] *
                      xShares[ Group == 1, , drop = FALSE ] )/
                  sum( xShares[ Group == 1, , drop = FALSE ] )
  XBetaEffect <- colSums( allCoef[ -xPos ] *
                          mXVal[ -xPos, , drop = FALSE ] ) + DEffect
  # effect
  effeG <- exp( XBetaEffect[ yCat ] )/( sum( exp( XBetaEffect ) ) ) -
           exp( XBetaRef[ yCat ] )/( sum( exp( XBetaRef ) ) )
}
# partial derivative of E_{k,ml} w.r.t. all estimated coefficients
if( method == "binary" ){
  derivCoef <- rep( NA, nCoef )
  derivCoef[ -xPos ] = ( exp( XBetaEffect )/( 1 + exp( XBetaEffect ))^2 -
                         exp( XBetaRef )/( 1 + exp( XBetaRef ))^2 ) *
                         allXVal[ -xPos ]
  derivCoef[ xPos ] = exp( XBetaEffect )/( 1 + exp( XBetaEffect))^2 * DEffect -
                      exp( XBetaRef )/( 1 + exp( XBetaRef ))^2 * DRef
} else if( method == "MNL" ){
  derivCoef <- matrix( NA, nrow=pCoef, ncol=NCoef )
  for( p in 1:NCoef ){
    if( p == yCat ){
      derivCoef[ -xPos, p ] <-
        ( exp( XBetaEffect[ p ] ) *
        ( 1 + sum( exp( XBetaEffect[ -yCat ] ) ) )/
        ( 1 + sum( exp( XBetaEffect ) ) )^2 -
          exp( XBetaRef[ p ] ) *
        ( 1 + sum( exp( XBetaRef[ -yCat ] ) ) )/
        ( 1 + sum( exp( XBetaRef ) ) )^2 ) * allXVal[ -xPos ]
```

```
      derivCoef[ xPos, p ] <-
        ( exp( XBetaEffect[ p ] ) *
        ( 1 + sum( exp( XBetaEffect[ -yCat ] ) ) )/
        ( 1 + sum( exp( XBetaEffect ) ) )^2 ) * DEffect -
        ( exp( XBetaRef[ p ] ) *
        ( 1 + sum( exp( XBetaRef[ -yCat ] ) ) )/
        ( 1 + sum( exp( XBetaRef ) ) )^2 ) * DRef
    } else{
      derivCoef[ -xPos, p ] <-
        ( ( exp( XBetaRef[ yCat ] ) * exp( XBetaRef[ p ] ) )/
          ( 1 + sum( exp( XBetaRef ) ) )^2 -
          ( exp( XBetaEffect[ yCat ] ) * exp( XBetaEffect[ p ] ) )/
          ( 1 + sum( exp( XBetaEffect ) ) )^2 ) * allXVal[ -xPos ]
      derivCoef[ xPos, p ] <-
          ( ( exp( XBetaRef[ yCat ] ) * exp( XBetaRef[ p ] ) )/
            ( 1 + sum( exp( XBetaRef ) ) )^2 ) * DRef -
          ( ( exp( XBetaEffect[ yCat ] ) * exp( XBetaEffect[ p ] ) )/
            ( 1 + sum( exp( XBetaEffect ) ) )^2 ) * DEffect
    }
  }
  derivCoef <- c( derivCoef )
} else{
  derivCoef <- rep( NA, nCoef )
  derivCoef[ -xPos ] = ( exp( XBetaEffect[ yCat ] ) * mXVal[ -xPos, yCat ] *
                         sum( exp( XBetaEffect ) ) -
                         exp( XBetaEffect[ yCat ] ) * sum( exp( XBetaEffect ) *
                         mXVal[ -xPos, ] ) )/
                       ( sum( exp( XBetaEffect ) ) )^2 -
                       ( exp( XBetaRef[ yCat ] ) * mXVal[ -xPos, yCat ] *
                         sum( exp( XBetaRef ) ) -
                         exp( XBetaRef[ yCat ] ) * sum( exp( XBetaRef ) *
                         mXVal[ -xPos, ] ) )/
                       ( sum( exp( XBetaRef ) ) )^2
  derivCoef[ xPos ] =  ( exp( XBetaEffect[ yCat ] ) * DEffect[ yCat ] *
                         sum( exp( XBetaEffect ) ) -
                         exp( XBetaEffect[ yCat ] ) *
                         sum( exp( XBetaEffect ) * DEffect[ yCat ] ) )/
                       ( sum( exp( XBetaEffect ) ) )^2 -
                       ( exp( XBetaRef[ yCat ] ) * DRef[ yCat ] *
                         sum( exp( XBetaRef ) ) -
                         exp( XBetaRef[ yCat ] ) *
                         sum( exp( XBetaRef ) * DRef[ yCat ] ) )/
                       ( sum( exp( XBetaRef ) ) )^2
}
# variance covariance of the coefficients (covariances set to zero)
vcovCoef <- diag( allCoefSE^2 )
```

```
  # approximate standard error of the effect
  effeGSE <- drop( sqrt( t( derivCoef ) %*% vcovCoef %*% derivCoef ) )
  result <- c( effect = effeG, stdEr = effeGSE )
  return( result )
}
```

where argument `allCoef` $= (\beta_0, \ldots, \beta_{k-1}, \beta_{k+1}, \ldots, \beta_K, \delta_1, \ldots, \delta_M)^\top$ are the coefficients of the binary or conditional logit regression or `allCoef` $= (\beta_{0,1}, \ldots, \beta_{k-1,1}, \beta_{k+1,1}, \ldots, \beta_{K,1}, \delta_{1,1}, \ldots, \delta_{M,1}, \beta_{0,P}, \ldots$ a vector of all the P sets of coefficients from the multi-nomial logit regression which does **not** include any values for the reference category, **with the coefficient of the reference group of the kth variable set to zero**; argument `allXVal` $= (x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_K, s_1, \ldots, s_M)^\top$ are the mean values of the respective explanatory variables and the shares of each of the categories of the kth variable **which includes the share of the reference group of the kth variable** or `allXVal` $= (z_{0,1}, \ldots, z_{t-1,1}, z_{t+1,1}, \ldots, z_{T,1}, s_{1,1}, \ldots, s_{M,1}, \ldots, z_{0,P}, \ldots, z_{t-1,P}, z_{t+1,P}, \ldots, z_{T,P}, s_{1,P}, \ldots, s_{M,P})^\top$ are the mean values of the P sets of explanatory variables and the P sets of shares of the tth variable **which includes the share of the reference group of the tth variable**; argument `xPos` is a vector of at least two elements indicating the position of the kth or tth variable in arguments `allCoef` and `allXVal`; argument `Group` is a vector of at least two elements consisting of -1s, 0s, and 1s, where a -1 indicates that the category belongs to the new reference category, a 1 indicates that the category belongs to the new category of interest, and a zero indicates that the category belongs to neither; argument `allCoefSE` $= (\text{se}(\beta_0), \ldots, \text{se}(\beta_{k-1}), \text{se}(\beta_{k+1}), \ldots, \text{se}(\beta_K), \text{se}(\delta_1), \ldots, \text{se}(\delta_M))^\top$ are the standard errors of all coefficients of the binary or conditional logit regression or `allCoefSE` $= (\text{se}(\beta_{0,1}), \ldots, \text{se}(\beta_{k-1,1}), \text{se}(\beta_{k+1,1}), \ldots, \text{se}(\beta_{K,1}), \text{se}(\delta_{1,1}), \ldots, \text{se}(\delta_{M,1}), \text{se}(\beta_{0,P}), \ldots, \text{se}(\beta_{k-1,P}), \text{se}(\beta_{k+1,P}), \ldots$ of the multi-nomial logit, respectively, **where the standard error for the reference group is included as zero**; argument `method = "binary", "MNL", "CondL"` identifies the estimator chosen, where `"binary"` indicates the binary logit estimator, `"MNL"` indicates the multi-nomial logit estimator, and `"CondL"` indicates the conditional logit; `yCat` is only relevant in connection with method `"MNL"` or `"CondL"` and indicates the pth output category of the multi-nomial logit that is of interest. Elements of arguments `allCoef`, `allXVal`, `Group`, and `allCoefSE` that belong to a category that is neither in the reference category nor in the category of interest, i.e., categories m with $D_{mr} = D_{ml} = 0$, can be omitted but the ommission must be consistent for all four arguments.

```
# Example
eff10a <- logEffGr( allCoef = c( 0.28, 0.003, 0.0075, 0, -0.034,
                                 -0.005, 0.89, -1.2 ),
                    allXVal = c( 1, 0.1, 0.3, 0.25, 0.15, 0.2, 2.34, 10.8 ),
                    xPos = c( 2:6 ), Group = c( 0, -1, -1, 1, 1 ) )
eff10a

##        effect         stdEr
## -5.339996e-07            NA

eff10b <- logEffGr( allCoef = c( 0.28, 0.003, 0.0075, 0, -0.034,
                                 -0.005, 0.89, -1.2 ),
```

```
                    allXVal = c( 1, 0.1, 0.3, 0.25, 0.15, 0.2, 2.34, 10.8 ),
                    xPos = c( 2:6 ), Group = c( 0, -1, -1, 1, 1 ),
                    allCoefSE = c( 0.03, 0.0001, 0.005, 0, 0.01,
                                   0.004, 0.05, 0.8 ) )
eff10b

##        effect         stdEr
## -5.339996e-07  4.613867e-06

# Example
eff11a <- logEffGr( allCoef = c( 0.28, 0.003, 0.0075, 0, -0.034,
                                 -0.005, 0.89, 0, 0.005, 0.06, 1.7, 0 ),
                    allXVal = c( 1, 0.5, 0.3, 0.2 ), xPos = c( 2:4 ),
                    Group = c( -1, -1, 1 ), yCat = 2, method = "MNL" )
eff11a

##     effect      stdEr
## -0.0131091         NA

eff11b <- logEffGr( allCoef = c( 0.28, 0.003, 0.0075, 0, -0.034,
                                 -0.005, 0.89, 0, 0.005, 0.06, 1.7, 0 ),
                    allXVal = c( 1, 0.5, 0.3, 0.2 ), xPos = c( 2:4 ),
                    Group = c( -1, -1, 1 ), yCat = 2, method = "MNL",
                    allCoefSE = c( 0.03, 0.0001, 0.005, 0, 0.01, 0.004,
                                   0.05, 0, 0.004, 0.5, 0.0078, 0 ) )
eff11b

##       effect        stdEr
## -0.013109100  0.003044428

# Example
eff12a <- logEffGr( allCoef = c( 0.28, 0.003, 0.0075, 0 ),
                    allXVal = c( 1, 0.5, 0.3, 0.2, 1, 0.4, 0.4, 0.1 ),
                    xPos = c( 2:4 ),
                    Group = c( -1, -1, 1 ), yCat = 2, method = "CondL" )
eff12a

##       effect        stdEr
## -7.03125e-05           NA

eff12b <- logEffGr( allCoef = c( 0.28, 0.003, 0.0075, 0 ),
                    allXVal = c( 1, 0.5, 0.3, 0.2, 1, 0.4, 0.4, 0.1 ),
                    xPos = c( 2:4 ),
                    allCoefSE = c( 0.03, 0.0001, 0.005, 0 ),
                    Group = c( -1, -1, 1 ), yCat = 2, method = "CondL" )
eff12b

##       effect        stdEr
## -7.03125e-05  1.23272e-21
```

APPENDIX

A. Additional helper functions

The following code defines a helper function that checks argument `xPos` of some functions that are defined above:

```
checkXPos <- function( xPos, minLength, maxLength, minVal, maxVal,
  requiredVal = NA ) {
  if( any( xPos != round( xPos ) ) ) {
    stop( "argument 'xPos' must be a vector of integers" )
  }
  if( length( xPos ) < minLength ) {
    stop( "argument 'xPos' must have a length equal to or larger than ",
      minLength )
  }
  if( length( xPos ) > maxLength ) {
    stop( "argument 'xPos' must have a length smaller than or equal to ",
      maxLength )
  }
  if( any( xPos < minVal ) ) {
    stop( "all elements of argument 'xPos' must be equal to or larger than ",
      minVal )
  }
  if( any( xPos > maxVal ) ) {
    stop( "all elements of argument 'xPos' must be smaller than or equal to ",
      maxVal )
  }
  if( max( table( xPos ) ) > 1 ) {
    stop( "all elements of argument 'xPos' may only occur once" )
  }
  if( !is.na( requiredVal ) ) {
    if( sum( xPos == requiredVal ) != 1 ) {
      stop( "argument 'xPos' must have exactly one element that is ",
        requiredVal )
    }
  }
}
```

The following code defines a helper function that checks whether $\beta_0 + \sum_{j=1}^{K} \beta_j x_j$ has a plausible value:

```
checkXBeta <- function( xBeta ) {
  if( any( abs( xBeta ) > 3.5 ) ) {
    warning( "At least one x'beta has an implausible value: ",
      paste( xBeta, collapse = ", " ) )
```

```
    }
}
```