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

Geraldine Henningsen & Arne Henningsen

January 28, 2021

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

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

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 \( k \)th (continuous) explanatory variable is:

\[ \epsilon_k = \frac{\partial E[y]}{\partial x_k} x_k = \beta_k \cdot x_k. \]

This semi-elasticity can be interpreted as: if the \( k \)th explanatory variable increases by one percent, the probability that \( y = 1 \) increases by \( \epsilon_k \) 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:

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

¹The 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:

\[
\text{se} (\epsilon_k) = \sqrt{\left( \frac{\partial \epsilon_k}{\partial (\beta_k, \beta_{k+1})} \right)^\top \text{Var} (\beta_k, \beta_{k+1}) \left( \frac{\partial \epsilon_k}{\partial (\beta_k, \beta_{k+1})} \right)}
\]

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

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 \( \beta_k \) and \( \beta_{k+1} \) is usually unknown. One can approximate equation (7) by assuming that the covariance between the estimates of \( \beta_k \) and \( \beta_{k+1} \), i.e., the off-diagonal element(s) of \( \text{Var} (\beta_k, \beta_{k+1}) \), is zero:

\[
\text{Var} (\beta_k, \beta_{k+1}) \approx \begin{bmatrix} \text{se} (\beta_k)^2 & 0 \\ 0 & \text{se} (\beta_{k+1})^2 \end{bmatrix}
\]

where \( \text{se} (\beta_k) \) and \( \text{se} (\beta_{k+1}) \) are the standard errors of the estimates of \( \beta_k \) and \( \beta_{k+1} \), 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:

\[
\text{se} (\epsilon_k) = \text{se} (\beta_k) \cdot x_k
\]

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

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

### Example

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

# 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 \( k \)th 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 \( x_k \):

\[
\epsilon_k \equiv \frac{\partial E[y]}{\partial x_k} x_k
\]  

(10)
can be approximated ‘around’ each ‘inner’ interval boundary \( b_1, \ldots, b_{M-1} \) by:

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

(11)

\[
\epsilon_{km} = \frac{\delta_{m+1} - \delta_m}{b_{m+1} - b_{m-1}} \cdot b_m
\]  

(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:

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

(13)

we get:

\[
\epsilon_{km} \approx \frac{\delta_{m+1} - \delta_m}{\frac{1}{2} (b_{m+1} + b_{m+1}) - \frac{1}{2} (b_{m-1} + b_m)} \cdot b_m
\]  

(14)

\[
= 2 \cdot \frac{\delta_{m+1} - \delta_m}{b_{m+1} - b_{m-1}} \cdot b_m
\]  

(15)

\( \forall \ m = 1, \ldots, M-1 \).

If \( b_M \) is infinity, we can assume:

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

(16)

\[
= 2 b_{M-1} - b_{M-2}
\]  

(17)

\[
= \frac{1}{2} (4 b_{M-1} - 2 b_{M-2})
\]  

(18)

\[
= \frac{1}{2} (b_{M-1} + 3 b_{M-1} - 2 b_{M-2}),
\]  

(19)

which is equivalent to setting:

\[
b_M = 3 b_{M-1} - 2 b_{M-2}.
\]  

(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:

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

where $s_m$ is the proportion of observations that are in the $m$th 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:

$$\epsilon_k \approx \sum_{m=1}^{M-1} w_m \epsilon_{km} \quad (22)$$

$$= (2s_1 + s_2) \frac{\delta_2 - \delta_1}{b_2 - b_0} b_1$$

$$+ \sum_{m=2}^{M-2} (s_m + s_{m+1}) \frac{\delta_{m+1} - \delta_m}{b_{m+1} - b_{m-1}} b_m$$

$$+ (s_{M-1} + 2s_M) \frac{\delta_M - \delta_{M-1}}{b_M - b_{M-2}} b_{M-1}. \quad (23)$$

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:

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

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

5
The $n$th element of the vector of partial derivatives $\partial \epsilon_k / \partial \delta$ can be obtained by:

\[
\frac{\partial \epsilon_k}{\partial \delta_n} = \sum_{m=1}^{M-1} w_m \frac{\partial \epsilon_{km}}{\partial \delta_n} = \left\{ \begin{array}{ll}
w_1 \frac{\partial \epsilon_{k1}}{\partial \delta_n} & \text{if } n = 1 \\
w_{n-1} \frac{\partial \epsilon_{k,n-1}}{\partial \delta_n} + w_n \frac{\partial \epsilon_{kn}}{\partial \delta_n} & \text{if } 2 \leq n \leq M - 1 \\
w_{M-1} \frac{\partial \epsilon_{k,M-1}}{\partial \delta_n} & \text{if } n = M \\
-2 \frac{w_1 b_1}{b_2 - b_0} & \text{if } n = 1 \\
2 w_{n-1} \frac{b_{n-1}}{b_n - b_{n-2}} - 2 w_n \frac{b_n}{b_{n+1} - b_{n-1}} & \text{if } 2 \leq n \leq M - 1 \\
2 w_{M-1} \frac{b_{M-1}}{b_M - b_{M-2}} & \text{if } n = M 
\end{array} \right.
\]

(25)

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

```r
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:

```r
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( bMInf <= 0 ) {
    bM <- bMInf
  }
  return( xBound )
}
```
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 $xBound = b = (b_0, ..., b_M)^\top$ 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 $\epsilon_k$ according to equations (15) and (22) and the respective standard error according to equations (24) and (27):

```r
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] <-
derivCoef[nInt] <-
if( nInt > 2 ) {
   for( n in 2:( nInt-1 ) ) {
      derivCoef[n] <-
      2 * weights[n-1] * xBound[n] / ( xBound[n+1] - xBound[n-1] ) -
   }
}
# variance-covariance matrix of the coefficients
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 = \( \delta = (se(\delta_1), \ldots, se(\delta_M))^\top \).

# 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
## 0.03260000 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 \( m \):

\[
E_{k,ml} = \mathbb{E}[y | b_{m-1} < x_k \leq b_m, x_{-k}] - \mathbb{E}[y | b_{l-1} < x_k \leq b_l, x_{-k}] 
= \beta_0 + \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j + \beta_k \mathbb{E}[x_k | b_{m-1} < x_k \leq b_m] 
- \beta_0 - \sum_{j \in \{1,\ldots,K\} \setminus k} \beta_j x_j - \beta_k \mathbb{E}[x_k | b_{l-1} < x_k \leq b_l] 
= \beta_k (\bar{x}_{km} - \bar{x}_{kl}), \tag{31}\]

where \( x_{-k} = (x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_K)^\top \) 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

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

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:

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

with

\[
\bar{x}_{km}^2 \equiv \mathbb{E}[x_k^2 | b_{m-1} < x_k \leq b_m] \forall \ m = 1, \ldots, M. \tag{37}\]
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 $1/(b_m - b_{m-1})$:

\[
x_{km}^2 \approx \int_{b_{m-1}}^{b_m} x_k^2 \frac{1}{b_m - b_{m-1}} \, dx_k
\]

\[
= \frac{1}{3} x_k^3 \frac{1}{b_m - b_{m-1}} |_{b_{m-1}}^{b_m}
\]

\[
= \frac{1}{3} b_m^3 \frac{1}{b_m - b_{m-1}} - \frac{1}{3} b_{m-1}^3 \frac{1}{b_m - b_{m-1}}
\]

\[
= \frac{b_m^3 - b_{m-1}^3}{3(b_m - b_{m-1})}.
\]

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:

\[
\text{se} (E_{k,ml}) = \sqrt{\left( \frac{\partial E_{k,ml}}{\partial \beta_k} \right) \text{Var} \left( \beta_k \beta_{k+1} \right) \left( \frac{\partial E_{k,ml}}{\partial \beta_k} \right)^\top},
\]

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

\[
\frac{\partial E_{k,ml}}{\partial \beta_k} = \bar{x}_{km} - \bar{x}_{kl}
\]

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:

\[
\frac{\partial E_{k,ml}}{\partial \beta_{k+1}} = \bar{x}_{km}^2 - \bar{x}_{kl}^2
\]

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

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

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

with arguments $\text{lowerBound} = b_{m-1}$ and $\text{upperBound} = b_m$ the lower and upper boundaries of the interval, respectively.
Using helper functions \textit{EXSquared} (equation 41) and \textit{elaIntBounds} (checking arguments \texttt{refBound} and \texttt{intBound}), the following function calculates the effect $E_{k,ml}$ and its approximate standard error $se(E)$ according to equations (31), (33), (36), and (42) to (44):

\begin{verbatim}
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 $E_{\cdot,k,ml}$
  eff <- xCoef[1] * xDiff + xCoef[2] * xSquaredDiff
  # partial derivative of $E_{\cdot,k,ml}$ 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 )
}
\end{verbatim}

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

\begin{verbatim}
# Example
# model equation that is linear in $x_k$
eff1a <- linEffInt( 0.4, refBound = c( 19, 34 ), intBound = c( 35, 55 ) )
eff1a
\end{verbatim}
# Example

## model equation that is quadratic in $x_k$

```
eff2a <- linEffInt(c( 0.4, -0.0003 ),
                refBound = c( 19, 34 ), intBound = c( 35, 55 ),
                xCoefSE = c( 0.002, 0.000001 )
)
eff2a
```

```
eff2b <- linEffInt( c( 0.4, refBound = c( 19, 34 ), intBound = c( 35, 55 ),
                xCoefSE = 0.03 )
eff1b
```

```
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

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

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

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

\[ - \beta_0 - \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j - \sum_{m=1}^{M} \delta_m E[D_m|x_k \in c_r^*] \]

\[ = \sum_{m=1}^{M} \delta_m (E[D_m|x_k \in c_l^*] - E[D_m|x_k \in c_r^*]) \] (51)

\[ = \sum_{m=1}^{M} \delta_m (D_{ml} - D_{mr}) \] (52)

with

\[ D_{mn} \equiv E[D_m|x_k \in c_n^*] \]

\[ = \frac{P(D_m = 1 \cap x_k \in c_n^*)}{P(x_k \in c_n^*)} \] (53)

\[ = \frac{E[D_m \mathbb{1}_{x_k \in c_n^*}]}{P(x_k \in c_n^*)} \] (54)

\[ = \frac{s_m v_{nm}}{\sum_{k=1}^{M} s_k v_{nk}} \] (55)

\[ \forall m = 1, \ldots, M; n \in \{r, l\}, \]

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:

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

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

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 \( \delta_1 = 0 \), 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 \( v_l = (1, 1, 0, 0, 0)^\top \):

\[
E_{k,\{1,2\}\{3,4\}} = \delta_1 \left( \frac{s_1}{s_1 + s_2} - \frac{0}{s_3 + s_4} \right) + \delta_2 \left( \frac{s_2}{s_1 + s_2} - \frac{0}{s_3 + s_4} \right) + \delta_3 \left( \frac{0}{s_1 + s_2} - \frac{s_3}{s_3 + s_4} \right) + \delta_4 \left( \frac{0}{s_1 + s_2} - \frac{s_4}{s_3 + s_4} \right) + \delta_5 \left( \frac{0}{s_1 + s_2} - \frac{0}{s_3 + s_4} \right) = \frac{s_2 \cdot \delta_2}{s_1 + s_2} - \frac{s_3 \cdot \delta_3}{s_3 + s_4} - \frac{s_4 \cdot \delta_4}{s_3 + s_4}.
\]

with approximate standard error:

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

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

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

with argument \( \text{xCoef} = (\delta_1, \ldots, \delta_M)^\top \) a vector of coefficients of the categorical variable of interest, where the coefficient of the reference group is set to 0, argument \( \text{xShares} \)
= \left( s_1, \ldots, s_M \right)^\top \) a vector of the corresponding shares of each category, argument \textbf{Group} = (D_{1l} - D_{1r}, \ldots, D_{Ml} - D_{Mr})^\top a vector consisting of $-1$s, $0$s, and $1$s, 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 \textbf{xCoefSE} = (se(\delta_1), \ldots, se(\delta_M))^\top, \textbf{where the standard error of the coefficient of the reference group is set to 0.} Elements of arguments \textbf{xCoef}, \textbf{xShares}, \textbf{Group}, and \textbf{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 omission must be consistent for all four arguments.

## Example:
```r
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
```

```r
## effect stdEr
## 0.172 NA
```

## Example:
```r
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
```

```r
## effect stdEr
## 0.172000000 0.001738489
```

## Example:
```r
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
```

```r
## effect stdEr
## 0.172 NA
```

## Example:
```r
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
```

```r
## 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:

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

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 \( k \)th (continuous) explanatory variable is:

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

where \( \phi(\cdot) \) is the probability density function of the standard normal distribution. The interpretation of the semi-elasticity \( \epsilon_k \) is identical to the one described in section 1.1. The value of the semi-elasticity \( \epsilon_k \) 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:

\[ \Pr(y = p) = \Phi \left( \mu_p - \sum_{j=1}^{K} \beta_j x_j \right) - \Phi \left( \mu_{p-1} - \sum_{j=1}^{K} \beta_j x_j \right) \quad \forall \ p \in \{1, \ldots, P\}, \]

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:

\[ y^* = \begin{cases} 
0 & \text{if } y \in \{1, \ldots, p^* - 1\} \\
1 & \text{if } y \in \{p^*, \ldots, P\}
\end{cases} \]
with \( p^* \in \{2, \ldots, P\} \). This reduces the ordered probit model to a binary probit model:

\[
\Pr(y^* = 1) = \Pr(y \in \{p^*, \ldots, P\})
\]

\[
= \sum_{p=p^*}^{P} \Pr(y = p)
\]

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

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

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

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

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

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

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., \( \beta_0 = -\mu_{p^*-1} \). 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:

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

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

\[ \text{If the ordered probit model is estimated with intercept, say, } \beta_0^* \text{ and (for identification) by normalising the first (internal) break point to zero, i.e., } \mu_1 = 0, \text{ the ordered probit model can be simplified to a binary probit model with intercept } \beta_0 = \beta_0^* - \mu_{p^*-1}. \]
estimated covariances rather than setting them to zero). In contrast, our simulations indicated that simplifying the calculation of the standard errors by assuming that \( \phi_0 + \sum_{j=1}^{K} \beta_j x_j \) is a constant, i.e., assuming \( \partial \phi_0 + \sum_{j=1}^{K} \beta_j x_j \partial \beta_n = 0 \forall n = 0, \ldots, K \), 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:

\[
\frac{\partial \epsilon_k}{\partial \beta_k} = \phi_0 + \sum_{j=1}^{K} \beta_j x_j \cdot x_k \tag{75}
\]

and an approximate standard error of \( \epsilon_k \) can be obtained by:

\[
\text{se} (\epsilon_k) = \sqrt{\phi_0 + \sum_{j=1}^{K} \beta_j x_j \cdot \text{Var} (\beta_k) \cdot \phi_0 + \sum_{j=1}^{K} \beta_j x_j \cdot x_k} \tag{76}
\]

\[
\text{se} (\epsilon_k) = \phi_0 + \sum_{j=1}^{K} \beta_j x_j \cdot x_k \cdot \text{se} (\beta_k), \tag{77}
\]

where \( \text{se} (\beta_k) \) 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:

\[
\frac{\partial \epsilon_k}{\partial (\beta_k \beta_{k+1})} = \left( \phi_0 + \sum_{j=1}^{K} \beta_j x_j \right) \cdot x_k \cdot \phi_0 + \sum_{j=1}^{K} \beta_j x_j \cdot 2x_k^2 \tag{78}
\]

and an approximate standard error can be calculated by:

\[
\text{se} (\epsilon_k) = \sqrt{\left( \frac{\partial \epsilon_k}{\partial \beta_k} \right)^\top \cdot \text{Var} \left( \begin{array}{cc} \beta_k & 0 \\ 0 & \beta_{k+1} \end{array} \right) \cdot \left( \frac{\partial \epsilon_k}{\partial \beta_{k+1}} \right)}, \tag{79}
\]

Using the helper functions \texttt{checkXPos} (checking argument \( \text{xPos} \), see appendix A) and \texttt{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, \( \epsilon_k \), and the corresponding standard error according to equations (62), (74), (78), and (79):

\begin{verbatim}
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" )
  }

  for( k in 1:nCoef ) {
    checkXPos( allCoef[ k ] + allXVal[ k ] * xPos )
    checkXBeta( allCoef[ k ] )
    seCoef <- se( allCoefSE[ k ] )
    # Calculate semi-elasticity
    elasK <- allCoef[ k ] * allXVal[ k ]
    # Calculate standard error
    seElas <- seCoef * allCoefSE[ k ]
    # Output
    print( elasK )
    print( seElas )
  }
}
\end{verbatim}
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 = (β0, ..., βK)⊤, or alternatively in the case of an ordered probit model allCoef = (−µm∗−1, β1, ..., βK)⊤, a vector of all coefficients; argument allXVal = (1, x1, ..., xK)⊤ 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)⊤ 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 = (se (β0), ..., se (βK))⊤ or alternatively in the case of an ordered probit model allCoefSE = (se (µm∗−1), se (β1), ..., se (βK))⊤, 4 the standard errors of the coefficients.

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

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

4Note that se (−µm∗−1) = \sqrt{\text{Var} (−µm∗−1)} = \sqrt{(∂(−µm∗−1)/∂µm∗−1)^2 \text{Var} (µm∗−1)} = \sqrt{(−1)^2 \text{Var} [µm∗−1] = \sqrt{\text{Var} (µm∗−1)} = se(µm∗−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 \( k \)th explanatory variable is interval-coded:

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

\[
D_m = \begin{cases} 
1 & \text{if } b_{m-1} < x_k \leq b_m \\
0 & \text{otherwise} 
\end{cases} \quad \forall m = 1, \ldots, M,
\]  
(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):

\[
\epsilon_k \equiv \frac{\partial E[y]}{\partial x_k} x_k \approx \sum_{m=1}^{M-1} w_m \epsilon_{km}
\]  
(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 \( \epsilon_{km} \) 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:

\[
\epsilon_{km} \approx \frac{E[y | b_{m} \leq x_{k} < b_{m+1}] - E[y | b_{m-1} \leq x_{k} < b_{m}]}{b_{m+1} - b_{m-1}} \quad \forall m = 1, \ldots, M - 1.
\]

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:

\[
\text{se} (\epsilon_{k}) = \sqrt{\frac{\partial \epsilon_{k}}{\partial (\beta^{\top} \delta^{\top})} \text{Var} \left( \frac{\partial \epsilon_{k}}{\partial (\beta^{\top} \delta^{\top})} \right)},
\]

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

\[
\frac{\partial \epsilon_{k}}{\partial (\beta^{\top} \delta^{\top})} = \sum_{m=1}^{M-1} u_{m} \frac{\partial \epsilon_{km}}{\partial (\beta^{\top} \delta^{\top})},
\]

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

\[
\begin{align*}
\frac{\partial \epsilon_{km}}{\partial \beta_{0}} & = \frac{2 (\phi_{m+1} - \phi_{m}) b_{m}}{b_{m+1} - b_{m-1}} \\
\frac{\partial \epsilon_{km}}{\partial \beta_{j}} & = \frac{2 (\phi_{m+1} - \phi_{m}) x_{j} b_{m}}{b_{m+1} - b_{m-1}} \quad \forall j \in \{1, \ldots, K\} \setminus k \\
\frac{\partial \epsilon_{km}}{\partial \delta_{m}} & = -\frac{2 \phi_{m} b_{m}}{b_{m+1} - b_{m-1}} \\
\frac{\partial \epsilon_{km}}{\partial \delta_{m+1}} & = \frac{2 \phi_{m+1} b_{m}}{b_{m+1} - b_{m-1}} \\
\frac{\partial \epsilon_{km}}{\partial \delta_{n}} & = 0 \quad \forall n \in \{1, \ldots, M\} \setminus \{m, m+1\}
\end{align*}
\]

with

\[
\phi_{m} = \phi \left( \beta_{0} + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_{j} x_{j} + \delta_{m} \right) \quad \forall m = 1, \ldots, M
\]

In the case of the ordered probit, one can simply replace \(\beta_{0}\) by \(-\mu_{p^{*}-1}\) in equation (92) (see section 2.1):

\[
\phi_{m} = \phi \left( -\mu_{p^{*}-1} + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_{j} x_{j} + \delta_{m} \right) \quad \forall m = 1, \ldots, M.
\]
Using the helper functions `elaIntWeights` (equation 21), `elaIntBounds` (equation 20), `checkXPos` (checking argument `xPos`, see appendix A), and `checkXBeta` (checking $\beta_0 + \sum_{j \in \{1, \ldots, K\}} \beta_j x_j + \sum_{m \in \{1, \ldots, M\}} \delta_m D_m$ 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):

```r
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 )
  }
}
```

22
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 coefficients
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 = ($\beta_0$, $\beta_{k-1}$, $\delta_1$, $\delta_{m^*-1}$, $\delta_{m^*+1}$, $\delta_M$, $\beta_{k+1}$, $\beta_K$)\textsuperscript{T}, or alternatively in the case of the ordered probit model allCoef = $(-\mu_{p^*-1}, \beta_1, \ldots, \beta_{k-1}$, $\delta_1, \ldots, \delta_{m^*-1}$, $\delta_{m^*+1}$, $\delta_M$, $\beta_{k+1}, \ldots, \beta_K$)\textsuperscript{T}, specifies the vector of all coefficients; argument allXVal = $(1, x_1, \ldots, x_{k-1}, s_1, \ldots, s_{m^*-1}, s_{m^*+1}, \ldots, s_M, x_{k+1}, \ldots, x_K)$\textsuperscript{T} 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 = $(b_0, \ldots, b_M)$\textsuperscript{T} indicates the boundaries of the intervals as in function linElaInt; and optional argument allCoefSE = $(\text{se}(\beta_0), \ldots, \text{se}(\beta_{k-1}), \text{se}(\delta_1), \ldots, \text{se}(\delta_{m^*-1}$),
se(δ_{m+1}),...,se(δ_M), se(β_{k+1}),...,se(β_K)^T, or alternatively in the case of an
ordered probit model allCoeffSE = (se(\mu_{m+1}), se(\beta_1),...,se(\beta_{k-1}), se(\delta_1),...,se(\delta_{m-1}),
se(\delta_{m+1}),...,se(\delta_M), se(\beta_{k+1}),...,se(\beta_K))^T, specifies the standard errors of all coefficients
in allCoeff.

# 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),
allCoeffSE = 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:

$$Pr(y = 1|x) = \Phi(\beta_0 + \sum_{j=1}^{K} \beta_jx_j),$$

(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:

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

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

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

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

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

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:

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

$$\cong \Phi \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k,k+1\}} \beta_j x_j \right. \hspace{1cm} (99)

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

$$- \Phi \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k,k+1\}} \beta_j x_j \right. \hspace{1cm} (100)

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

$$= \Phi \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k,k+1\}} \beta_j x_j + \beta_k \bar{x}_{km} + \beta_{k+1} \bar{x}_2{km} \right)$$

$$- \Phi \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k,k+1\}} \beta_j x_j + \beta_k \bar{x}_{kl} + \beta_{k+1} \bar{x}_2{kl} \right),$$

where $\bar{x}_{km} \equiv E[x_k|b_{m-1} < x_k \leq b_m]$ and $\bar{x}_2{km} \equiv E[x_k^2|b_{m-1} < x_k \leq b_m] \forall m = 1, \ldots, M$ remain the same as outlined in section 1.3.\(^6\) If the values of $\bar{x}_{km}$ and $\bar{x}_2{km}$ 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:

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

(101)

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

\begin{align}
\frac{\partial E_{k,ml}}{\partial \beta_0} &= \phi_m - \phi_l \\
\frac{\partial E_{k,ml}}{\partial \beta_j} &= (\phi_m - \phi_l)x_j \quad \forall \ j \in \{1, \ldots, k-1, k+2, \ldots, K\} \\
\frac{\partial E_{k,ml}}{\partial \beta_k} &= \phi_m \bar{x}_{km} - \phi_l \bar{x}_{kl} \\
\frac{\partial E_{k,ml}}{\partial \beta_{k+1}} &= \phi_m \bar{x}_{k+1,km} - \phi_l \bar{x}_{k+1,kl}
\end{align}

(102) \quad (103) \quad (104) \quad (105)

with $\phi_m \equiv \phi(\hat{\beta}_0 + \sum_{j=1}^{K} \beta_j x_j + \beta_k \bar{x}_{km} + \beta_{k+1} \bar{x}_{km}) \quad \forall m = 1, \ldots, M$.

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):

```r
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 = (\(\beta_0, \ldots, \beta_K\))^T, or alternatively for an ordered probit model allCoef = (\(-\mu_{p-1}, \beta_1, \ldots, \beta_K\))^T, is a vector containing all coefficients; argument allXVal = (1, x_1, \ldots, x_K)\^T is a vector containing all values of the corresponding explanatory variables; argument xPos = k + 1 or (k + 1, k + 2)^T 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 = (se(\(\hat{\beta}_0\)), \ldots, se(\(\hat{\beta}_K\)))\^T, or alternatively for ordered probit models allCoefSE = (se(\(\hat{\mu}_{p-1}\)), se(\(\hat{\beta}_1\)), \ldots, se(\(\hat{\beta}_K\)))\^T, 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

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

$$D_m = \begin{cases} 1 & \text{if } x_k \in c_m \\ 0 & \text{otherwise} \end{cases} \quad \forall m = 1, \ldots, M. \hspace{1cm} (107)$$

Like in section 1.4, the $k$th explanatory 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 probit regression, equation (52) modifies to:

$$E_{k,lr} = E[y|x_k \in c^*_l] - E[y|x_k \in c^*_r] \hspace{1cm} (108)$$

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

$$- \Phi \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j + \sum_{m=1}^M \delta_m E[D_m|x_k \in c^*_r] \right) \hspace{1cm} (110)$$

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

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

where $D_{ml}$ is defined as in equation (56). In the case of an ordered probit regression, $\beta_0$ 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:

$$se(E_{k,lr}) = \sqrt{\left( \frac{\partial E_{k,lr}}{\partial \beta_0} \right)^\top \cdot \text{Var} \left( \begin{pmatrix} \beta \\ \delta \end{pmatrix} \right) \cdot \frac{\partial E_{k,lr}}{\partial \beta_0}},$$  \hspace{1cm} (111)$$

with the partial derivatives defined as:

$$\frac{\partial E_{k,lr}}{\partial \beta_0} = \phi_{ml}(\cdot) - \phi_{mr}(\cdot) \hspace{1cm} (112)$$

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

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

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

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

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

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

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

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

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

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

```r
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 ] ) /
```

30
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 = (β₀, ..., β_{k-1}, δ₁, ..., δ_{M}, β_{k+1}, ..., β_{K})^T or alternatively allCoef = (-μ_{p-1}, β₁, ..., β_{k-1}, δ₁, ..., δ_{M}, β_{k+1}, ..., β_{K})^T 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., δ_{m*r} ≡ 0; argument allXVal = (x₁, ..., x_{k-1}, s₁, ..., s_{M}, x_{k+1}, ..., x_{K})^T 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*r}) and, equally, the positions of the shares s₁, ..., s_{M} in argument allXVal; argument Group = (D₁l - D₁r, ..., D_{M}l - D_{Mr})^T 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 = (se(β₀), ..., se(β_{k-1}), se(δ₁), ..., se(δ_{M}), se(β_{k+1}), ..., se(β_{K}))^T, or alternatively in the case of an ordered probit model allCoefSE = (se(μ_{p-1}), se(β₁), ..., se(β_{k-1}), se(δ₁), ..., se(δ_{M}), se(β_{k+1}), ..., se(β_{K}))^T 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., se(δ_{m*r}) ≡ 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 omission 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
## 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:

\[
Pr(y = 1|x) = \frac{\exp(\beta_0 + \sum_{k=1}^{K} \beta_k x_k)}{1 + \exp(\beta_0 + \sum_{k=1}^{K} \beta_k x_k)},
\]

(119)

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

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

\[
\epsilon_k = \frac{\partial Pr(y = 1)}{\partial x_k} \cdot \bar{x}_k = \frac{\exp(\beta_0 + \sum_{k=1}^{K} \beta_k \bar{x}_k)}{(1 + \exp(\beta_0 + \sum_{k=1}^{K} \beta_k \bar{x}_k))^2} \cdot \beta_k \bar{x}_k
\]

(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

\[
e_k = \frac{\exp(\beta_0 + \sum_{k \in \{1, \ldots, K\} \setminus k+1} \beta_k x_{k} + \beta_{k+1} x_{k}^2)}{\left(1 + \exp(\beta_0 + \sum_{k \in \{1, \ldots, K\} \setminus k+1} \beta_k x_{k} + \beta_{k+1} x_{k}^2)\right)^2} \cdot (\beta_k x_k + 2\beta_{k+1} x_{k}^2) \tag{121}
\]

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

\[
se(\epsilon_k) = \sqrt{\frac{\exp(\beta_0 + \sum_{k=1}^{K} \beta_k x_k)}{(1 + \exp(\beta_0 + \sum_{k=1}^{K} \beta_k x_k))^2} \cdot \bar{x}_k \cdot Var(\epsilon_k) \cdot \frac{\exp(\beta_0 + \sum_{k=1}^{K} \beta_k x_k)}{(1 + \exp(\beta_0 + \sum_{k=1}^{K} \beta_k x_k))^2} \cdot \bar{x}_k} \tag{122}
\]

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

\[
\frac{\partial \epsilon_k}{\partial \beta_k} = \left( \frac{\exp(\beta_0 + \sum_{k \in \{1, \ldots, K\} \setminus k+1} \beta_k x_{k} + \beta_{k+1} x_{k}^2)}{\left(1 + \exp(\beta_0 + \sum_{k \in \{1, \ldots, K\} \setminus k+1} \beta_k x_{k} + \beta_{k+1} x_{k}^2)\right)^2} \cdot \bar{x}_k \right) \tag{124}
\]

\[
se(\epsilon_k) = \sqrt{\left( \frac{\partial \epsilon_k}{\partial \beta_k} \right) \cdot Var(\epsilon_k) \cdot \frac{\partial \epsilon_k}{\partial \beta_k}} \tag{125}
\]

The multinomial logit model with continuous explanatory variables:

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

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

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

\[
e_{k,p} = \frac{\partial Pr(y = p)}{\partial x_k} \cdot \bar{x}_k
\]

\[
= \pi_p \left( \frac{\beta_{k,p}}{1 + \sum_{p \in \{1, \ldots, P\} \setminus p^*} \exp(\beta_{0,p} + \sum_{k=1}^{K} \beta_{k,p} x_{k})} + \sum_{o \in \{1, \ldots, P\} \setminus \{p,p^*\}} (\beta_{k,p} - \beta_{o,p}) \pi_o \right) \cdot \bar{x}_k \tag{127}
\]
This semi-elasticity can be interpreted as: if the \( k, p \)th explanatory variable increases 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,

\[
\Pr(y = p | x) = \pi_p = \frac{\exp(\beta_{0,p} + \sum_{k\in\{1, \ldots, K\}\setminus k+1} \beta_{k,p}x_k + \beta_{k+1,p}x_k^2)}{1 + \sum_{p\in\{1, \ldots, P\}\setminus p'} \exp(\beta_{0,p} + \sum_{k\in\{1, \ldots, K\}\setminus k+1} \beta_{k,p}x_k + \beta_{k+1,p}x_k^2)}
\] (128)
equation (127) modifies to

\[
\epsilon_{k,p} = \frac{\partial \Pr(y = p)}{\partial \bar{x}_k} = \pi_p \left( \frac{(\beta_{k,p} + 2\beta_{k+1,p}\bar{x}_k)}{[1 + \sum_{p\in\{1, \ldots, P\}\setminus p'} \exp(\beta_{0,p} + \sum_{k\in\{1, \ldots, K\}\setminus k+1} \beta_{k,p}x_k + \beta_{k+1,p}x_k^2)]^2} \right) \bar{x}_k
\] (129)

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

\[
\frac{\partial \epsilon_{k,p}}{\partial \beta_{k,p}} = \pi_p \left( \frac{\exp(\beta_{0,p} + \sum_{k=1}^K \beta_{k,p}x_k)}{[1 + \sum_{p\in\{1, \ldots, P\}\setminus p'} \exp(\beta_{0,p} + \sum_{k=1}^K \beta_{k,p}x_k)]^2} \bar{x}_k
\] (130)

\[
se(\epsilon_{k,p}) = \sqrt{\left( \frac{\partial \epsilon_{k,p}}{\partial \beta_{k,p}} \right)^\top \Var(\beta_{k,p}) \frac{\partial \epsilon_{k,p}}{\partial \beta_{k,p}}} = \sqrt{\frac{\partial \epsilon_{k,p}}{\partial \beta_{k,p}} se(\beta_{k,p})}
\] (131)
and in case of a quadratic covariate $\bar{x}^2_k$

$$
\frac{\partial \epsilon_{k,p}}{\partial (\beta_{k,p})} = \frac{\exp(\beta_0 + \sum_{k \in \{1,\ldots,K\}\backslash k+1} \beta_{k,p} \bar{x}_k + \beta_{k+1,p} \bar{x}^2_k))}{[1 + \sum_{p \in \{1,\ldots,P\}\backslash p^*} \exp(\beta_0 + \sum_{k \in \{1,\ldots,K\}\backslash k+1} \beta_{k,p} \bar{x}_k + \beta_{k+1,p} \bar{x}^2_k)]^2 \bar{x}_k} + \pi_p \sum_{o \in \{1,\ldots,P\}\backslash \{p,p^*\}} ((\beta_{k,p} + 2\beta_{k+1,p} \bar{x}_k) - (\beta_{k,o} + 2\beta_{k+1,0} \bar{x}_k)) \pi_o \cdot \bar{x}_k
$$

$$
\frac{\partial \epsilon_{k,p}}{\partial (\beta_{k+1,p})} = \frac{\exp(\beta_0 + \sum_{k \in \{1,\ldots,K\}\backslash k+1} \beta_{k,p} \bar{x}_k + \beta_{k+1,p} \bar{x}^2_k))}{[1 + \sum_{p \in \{1,\ldots,P\}\backslash p^*} \exp(\beta_0 + \sum_{k \in \{1,\ldots,K\}\backslash k+1} \beta_{k,p} \bar{x}_k + \beta_{k+1,p} \bar{x}^2_k)]^2 \bar{x}_k} + \pi_p \sum_{o \in \{1,\ldots,P\}\backslash \{p,p^*\}} ((\beta_{k,p} + 2\beta_{k+1,p} \bar{x}_k) - (\beta_{k,o} + 2\beta_{k+1,0} \bar{x}_k)) \pi_o \cdot \bar{x}_k
$$

$$
\text{se} (\epsilon_{k,p}) = \sqrt{\left( \frac{\partial \epsilon_{k,p}}{\partial (\beta_{k,p})} \right)^\top \cdot \text{Var} \left( \frac{\partial \epsilon_{k,p}}{\partial (\beta_{k,p})} \right) \cdot \frac{\partial \epsilon_{k,p}}{\partial (\beta_{k+1,p})}}
$$

(132)

Conditional logit model with continuous explanatory variables:

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

(134)

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

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

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

(135)

This semi-elasticity can be interpreted as: if the $t$th explanatory variable increases 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

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

(136)

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

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

(137)
and in case of a quadratic covariate $z_t^2$

$$\frac{\partial \epsilon_t}{\partial \gamma_t} = \frac{(\pi_p - \pi_p^2) z_t}{(\pi_p - \pi_p^2) 2 z_t^2}$$

(138)

$$\text{se}(\epsilon_t) = \sqrt{\begin{pmatrix} \frac{\partial \epsilon_t}{\partial \gamma_t} \\ \frac{\partial \epsilon_t}{\partial (\gamma_t^2)} \end{pmatrix}^\top \text{Var} \begin{pmatrix} \gamma_t \\ \gamma_t^2 \end{pmatrix} \frac{\partial \epsilon_t}{\partial \gamma_t}}$$

(139)

For the nested logit model with continuous explanatory variables at both the branch and the twig level, we have to differentiate between the probability of choice of the $o$th branch, $B_o$,

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

(140)

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

(141)

and the combined probability of choosing the $p$th alternative in the $o$th branch

$$\Pr(y = p, y \in B_o) = \Pr(y = p|x, y \in B_o) \cdot \Pr(y \in B_o|x) = \pi_p \cdot \pi_o$$

$$= \frac{\exp(\gamma_0 + \sum_{k=1}^{K} \beta_k / \lambda_o x_{k,p})}{\sum_{p \in B_o} \exp(\gamma_0 + \sum_{k=1}^{K} \beta_k / \lambda_o x_{k,p})} \cdot \frac{\exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o)}{\sum_{o=1}^{O} \exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o)}$$

(142)

where $z_o = (1, z_{1,o}, \ldots, z_{T,o})^\top$ are the branch specific sets of characters (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:

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

(143)

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

$$\epsilon_{t,o} = (\pi_o - \pi_o^2) \cdot (\gamma_t z_{t,o} + 2 \gamma_{t+1} z_{t,o}^2)$$

(144)

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:

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

(145)

and for the quadratic form

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

(146)

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

\[
se (\epsilon_{t,o}) = (\pi_o - \pi_o^2) \cdot \bar{z}_{t,o} \cdot se (\gamma_t)
\]

(147)

and in case of a quadratic covariate \( z_{t,o}^2 \)

\[
\frac{\partial \epsilon_{t,o}}{\partial (\gamma_t)} = \left( \begin{array}{c} \pi_o - \pi_o^2 \\ \pi_o - \pi_o^2 \end{array} \right) z_{t,o}
\]

(148)

\[
se (\epsilon_{t,o}) = \sqrt{\left( \begin{array}{c} \frac{\partial \epsilon_{t,o}}{\partial (\gamma_t)} \\ \frac{\partial \epsilon_{t,o}}{\partial (\gamma_t)} \end{array} \right) \cdot \text{Var} \left( \begin{array}{c} \gamma_t \\ \gamma_t \end{array} \right) \cdot \frac{\partial \epsilon_{t,o}}{\partial (\gamma_t)}}
\]

(149)

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

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

(150)

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

\[
\frac{\partial \epsilon_{k,p}}{\partial (\beta_k)} = \left( \begin{array}{c} \pi_o(\pi_p - \pi_p^2) \frac{1}{\lambda_o} + \pi_p^2(\pi_o - \pi_o^2) \lambda_o IV_o \\ \pi_o(\pi_p - \pi_p^2) \frac{1}{\lambda_o} + \pi_p^2(\pi_o - \pi_o^2) \lambda_o IV_o \end{array} \right) \cdot \bar{x}_{k,p}
\]

(151)

\[
se (\epsilon_{k,p}) = \sqrt{\left( \begin{array}{c} \frac{\partial \epsilon_{k,p}}{\partial (\beta_k)} \\ \frac{\partial \epsilon_{k,p}}{\partial (\beta_k)} \end{array} \right) \cdot \text{Var} \left( \begin{array}{c} \beta_k \\ \beta_{k+1} \end{array} \right) \cdot \frac{\partial \epsilon_{k,p}}{\partial (\beta_k)}}
\]

(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 approximate standard errors as defined in equations (135), (136), (137), and (139) for the conditional logit.
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" ){
    NCoeff <- length( allCoef )
    mCoeff <- matrix( allCoef, nrow = length( allXVal ) )
    nCoeff <- dim( mCoeff )[1]
    pCoeff <- dim( mCoeff )[2]
    # Checking standard errors
    if( NCoeff != length( allCoefSE ) ) {
      stop( "arguments 'allCoef' and 'allCoefSE' must have the same length" )
    }
  } else{
    nCoef <- length( allCoef )
    NCoeff <- 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 = NCoeff )
    nXValBra <- dim( mXValBra )[1]
    pXValBra <- dim( mXValBra )[2]
    if( NCoeff != nXValBra ){
      stop( "arguments 'allCoefBra' and 'allXValBra' must have the same length" )
    }
  }
}

checkXPos( xPos, minLength = 1, maxLength = 2, minVal = 1, maxVal = nCoeff )
# Check x values
if( method == "binary" || method == "MNL" ){
  if( nCoeff != length( allXVal ) ){
    stop( "arguments 'allCoef' and 'allXVal' must have the same length" )
  }
} else if( method == "CondL" ){
  mXValBra <- matrix( allXValBra, nrow = NCoeff )
  nXValBra <- dim( mXValBra )[1]
  pXValBra <- dim( mXValBra )[2]
  if( NCoeff != nXValBra ){
    stop( "arguments 'allCoefBra' and 'allXValBra' must have the same length" )
  }
}

38
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 )
  }
}

39
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" ){
    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" ){
    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{
    xVal <- rep( NA, pXVal )
    for( p in 1:pXVal ){
        xVal[p] <- mXVal[ xPos[ 1 ], p ]
    }
    coef <- matrix( NA, nrow = 0, 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 )
}
with

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

- argument \( \text{allXVal} = (1, \ldots, \bar{x}_K)\) a vector of all corresponding sample means and 1 for the intercept for the binary or multinomial logit, \( \text{allXVal} = (1, \ldots, \bar{x}_{T,1}, \ldots, 1, \ldots, \bar{x}_{T,P}) \) a vector of the \( P \) sets of explanatory variables from the conditional logit, or \( \text{allXVal} = ((1, \ldots, \bar{x}_{K,P,o}), \ldots, (1, \ldots, \bar{x}_{K,P,O}))\) 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 \( \text{allXVal} \);

- argument \( \text{allCoefSE} = (\text{se}(\beta_0), \ldots, \text{se}(\beta_K))\) 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 multinomial logit regression;

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

- argument \( \text{method} = \text{c("binary", "MNL", "CondL", "NestL")} \) 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 "\text{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 \( \text{allCoef} \) are already corrected by \( \beta_k \lambda_o \);

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

---

\(^8\)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),
line11 = c(1, 3.3, 4.5, 2.24, 0.1, 0.987),
line12 = 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),
line11 = 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),
line11 = c(1, 3.3, 3.3^2, 2.34, 0.1, 0.987),
line12 = 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),
line11 = 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),
line11 = c(1, 8.4, 0.06),
line12 = 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

43
# 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
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

# 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

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

# 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
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" )

3.2. Semi-elasticities from interval-coded explanatory variables

Logit model, where the $k$th explanatory variable is interval-coded:

$$
\Pr(y = 1 | x) = \frac{\exp(\beta_0 + \sum_{j \in \{1, \ldots, K\} \backslash k} \beta_j x_j + \sum_{m \in \{1, \ldots, M\} \backslash m} \delta_m D_m)}{1 + \exp(\beta_0 + \sum_{j \in \{1, \ldots, K\} \backslash k} \beta_j x_j + \sum_{m \in \{1, \ldots, M\} \backslash m} \delta_m D_m)}, \quad (153)
$$

with

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

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

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

where

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

and

$$
\exp_{m}(\cdot) \equiv \exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \backslash k} \beta_j x_j + \delta_{m} \right). \quad (157)
$$

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

$$
\epsilon_{km,p} \approx 2 \left( \frac{\exp_{m+1,p}(\cdot)}{1 + \sum_{p \in \{1, \ldots, P\} \backslash p} \exp_{m+1,p}(\cdot)} - \frac{\exp_{m,p}(\cdot)}{1 + \sum_{p \in \{1, \ldots, P\} \backslash p} \exp_{m,p}(\cdot)} \right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}} \quad (158)
$$
where \( \exp_{m,p} \) is defined as
\[
\exp_{m,p} (\cdot) \equiv \exp \left( \beta_{0,p} + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_{j,p} x_j + \delta_{m,p} \right).
\]
and \( \exp_{m+1,p} \) accordingly.

For the conditional logit, equation (155) modifies to
\[
\epsilon_{tm,p} \approx 2 \left( \sum_{p \in C} \exp_{m+1,p} (\cdot) \right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}}.
\]
where \( \exp_{m,p} \) is defined as
\[
\exp_{m,p} (\cdot) \equiv \exp \left( \gamma_{0} + \sum_{j \in \{1, \ldots, T\} \setminus t} \gamma_j z_{j,p} + \delta_m \right).
\]
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:
\[
\text{se} \left( \epsilon_{km} \right) = \sqrt{\text{Var} \left( (\beta^\top \delta^\top)^\top \right)} \cdot \frac{\partial \epsilon_{km}}{\partial (\beta^\top \delta^\top)^\top}.
\]
where \( \epsilon_{km} \) indicates the (approximate) standard error of \( \epsilon_{km} \), \( \text{Var} \left( (\beta^\top \delta^\top)^\top \right) \) indicates the variance-covariance matrix of the estimates of the coefficient vector \( (\beta^\top \delta^\top)^\top \), and the gradient vector is:
\[
\frac{\partial \epsilon_{km}}{\partial (\beta^\top \delta^\top)^\top} = \sum_{m=1}^{M-1} w_m \frac{\partial \epsilon_{km}}{\partial (\beta^\top \delta^\top)^\top}.
\]
with the elements of the gradient vector \( \frac{\partial \epsilon_{km}}{\partial (\beta^\top \delta^\top)^\top} \):
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{align*}
\frac{\partial \epsilon_{km,p}}{\partial \beta_{0,p}} &= 2 \left( \frac{\exp_{m+1,p}(\cdot) \cdot (1 + \sum_{o \in \{1, \ldots, P\}} \exp_{m+1,o}(\cdot))}{[1 + \sum_{p \in \{1, \ldots, P\}} \exp_{m+1,p}(\cdot)]^2} \right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}} \\
\frac{\partial \epsilon_{km,p}}{\partial \beta_{0,o}} &= 2 \left( \frac{\exp_{m+1,o}(\cdot) \cdot (1 + \sum_{o \in \{1, \ldots, P\}} \exp_{m+1,o}(\cdot))}{[1 + \sum_{p \in \{1, \ldots, P\}} \exp_{m+1,o}(\cdot)]^2} \right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}} \quad \forall \; o \neq p \tag{169} \\
\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, \ldots, P\}} \exp_{m+1,o}(\cdot))}{[1 + \sum_{p \in \{1, \ldots, P\}} \exp_{m+1,p}(\cdot)]^2} \right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}} \quad \forall \; j \in \{1, \ldots, K\} \setminus k \tag{171} \\
\frac{\partial \epsilon_{km,p}}{\partial \beta_{j,o}} &= 2 \left( \frac{\exp_{m+1,o}(\cdot) \cdot x_j (1 + \sum_{o \in \{1, \ldots, P\}} \exp_{m+1,o}(\cdot))}{[1 + \sum_{p \in \{1, \ldots, P\}} \exp_{m+1,o}(\cdot)]^2} \right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}} \quad \forall \; o \neq p, \; j \in \{1, \ldots, K\} \setminus k \tag{172} \\
\frac{\partial \epsilon_{km}}{\partial \delta_{m,p}} &= -2 \frac{\exp_{m,p}(\cdot) \cdot (1 + \sum_{o \in \{1, \ldots, P\}} \exp_{m,o}(\cdot))}{[1 + \sum_{p \in \{1, \ldots, P\}} \exp_{m,p}(\cdot)]^2} \cdot \frac{b_m}{b_{m+1} - b_{m-1}} \tag{173} \\
\frac{\partial \epsilon_{km}}{\partial \delta_{m,o}} &= 2 \frac{\exp_{m,o}(\cdot)}{[1 + \sum_{p \in \{1, \ldots, 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, \ldots, P\}} \exp_{m+1,o}(\cdot))}{[1 + \sum_{p \in \{1, \ldots, 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, \ldots, 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}}{\partial \delta_{n,p}} &= 0 \; \forall \; n \in \{1, \ldots, M\} \setminus \{m, m + 1\} \tag{177} \\
\frac{\partial \epsilon_{km}}{\partial \delta_{n,o}} &= 0 \; \forall \; n \in \{1, \ldots, M\} \setminus \{m, m + 1\}, \; o \neq p \tag{178} 
\end{align*}
\]

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
\[
\frac{\partial \epsilon_{tm}}{\partial \gamma_0} = \frac{2}{\left(\sum_{p \in C} \exp \left[m_{p+1}(\cdot)\right]\right)^2} \left(\exp \left[m_{p+1}(\cdot)\right] \cdot \sum_{p \in C} \exp \left[m_{p}(\cdot)\right] - \exp \left[m_{p}(\cdot)\right] \cdot \sum_{p \in C} \exp \left[m_{p+1}(\cdot)\right]\right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}} = 0
\]

\[
\frac{\partial \epsilon_{tm}}{\partial \gamma_j} = \frac{2}{\left(\sum_{p \in C} \exp \left[m_{p+1}(\cdot)\right]\right)^2} \left(\exp \left[m_{p+1}(\cdot)\right] \cdot \sum_{p \in C} \exp \left[m_{p}(\cdot)\right] - \exp \left[m_{p}(\cdot)\right] \cdot \sum_{p \in C} \exp \left[m_{p+1}(\cdot)\right]\right) \cdot \frac{b_m}{b_{m+1} - b_{m-1}} \forall j \in \{1, \ldots, T\} \setminus t
\]

\[
\frac{\partial \epsilon_{tm}}{\partial \delta_m} = 0
\]

\[
\frac{\partial \epsilon_{tm}}{\partial \delta_{m+1}} = 0
\]

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

Using the helper functions \texttt{elaIntWeights} (equation 21), \texttt{elaIntBounds} (equation 20), \texttt{checkXPos} (checking argument \texttt{xPos}, see appendix A), and \texttt{checkXBeta} (checking \(\beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j + \sum_{m \in \{1, \ldots, M\} \setminus m} \delta_mD_m\) for plausible values, see appendix A), and function \texttt{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-nominal logit function, and equations (160) and (179) to (183) for the conditional logit, respectively:

\[
\text{logElaInt <- function}( \text{allCoef}, \text{allXVal}, \text{xPos}, \text{xBound}, yCat = \text{NA}, \text{allCoefSE} = \text{rep}(\text{NA}, \text{length( allCoef ) })),
\]

49
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 <- elaIntWeights( shareVec )

# calculation of the semi-elasticity
semEla <- linElaiInt( 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 ){
    gradExpVecO <- ( 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 * ( gradExpVecO[m] - gradExpVecO[m+1] ) * 
          allXVal[ -xPos ] * xBound[m+1] / ( xBound[m+2] - xBound[m] )
        gradM[ xPos[m], m, p ] <- - 2 * gradExpVecO[m] * xBound[m+1] / 
          ( xBound[m+2] - xBound[m] )
        gradM[ xPos[m+1], m, p ] <- 2 * gradExpVecO[m+1] * xBound[m+1] / 
          ( xBound[m+2] - xBound[m] )
      }
    }
  }
  gradM <- apply( gradM, 2, function( x ) x )
\[
\text{else} \{
\text{gradM} \leftarrow \text{matrix}(0, \text{nCoef}, \text{nInt} - 1)\)
\begin{align*}
\text{for}( & \text{m in 1:( nInt - 1 )}) \{ \\
& \text{gradM[ -xPos, m ] } \leftarrow 2 \ast \\
& \text{( ( exp( xBeta[ m+1, yCat ] ) } \ast \text{mXVal[ -xPos, yCat ] } \ast \\
& \text{sum( exp( xBeta[ m+1, ] ) ) ) } - \\
& \text{exp( xBeta[ m+1, yCat ] ) } \ast \\
& \text{rowSums( exp( xBeta[ m+1, ] ) } \ast \text{mXVal[ -xPos, , drop = FALSE ] ) } ) \} \big/ \\
& \text{( sum( exp( xBeta[ m+1, ] ) ) ) } \ast 2 - \\
& \text{( exp( xBeta[ m, yCat ] ) } \ast \text{mXVal[ -xPos, yCat ] } \ast \\
& \text{sum( exp( xBeta[ m, ] ) ) ) } - \\
& \text{exp( xBeta[ m, yCat ] ) } \ast \\
& \text{rowSums( exp( xBeta[ m, ] ) } \ast \text{mXVal[ -xPos, , drop = FALSE ] ) ) } \big/ \\
& \text{sum( exp( xBeta[ m, ] ) ) ) } \ast 2 \ast \\
& \text{xBound}[m+1] / ( xBound[m+2] - xBound[m] ) \\
\text{gradM[ xPos[m+1], m ] } \leftarrow 0 \\
\text{gradM[ xPos[m], m ] } \leftarrow 0
\}
\end{align*}
\]
\begin{itemize}
\item # partial derivative of the semi-elasticity
\item # w.r.t. all estimated coefficients
\end{itemize}
\text{derivCoef } \leftarrow \text{rep}(0, \text{length( allCoef )})
\begin{align*}
\text{for}( & \text{m in 1:( nInt - 1 )}) \{ \\
& \text{derivCoef } \leftarrow \text{derivCoef } + \text{weights[m] } \ast \text{gradM[,m]}
\}
\end{align*}
\begin{itemize}
\item # variance-covariance matrix of the coefficients
\end{itemize}
\text{vcovCoef } \leftarrow \text{diag( allCoefSE } \ast 2 )
\begin{itemize}
\item # standard error of the (average) semi-elasticity
\end{itemize}
\text{semElaSE } \leftarrow \text{drop( sqrt( t( derivCoef ) } \ast \text{vcovCoef } \ast \text{derivCoef } ) )}
\begin{itemize}
\item # prepare object that will be returned
\end{itemize}
\text{return( result )}
\]

where argument \text{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)^T, is a vector of all coefficients from the binary or conditional logit regression or \text{allCoef } = (\beta_0, \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, \ldots, \\
\beta_0, p, \ldots, \beta_{k-1,1}, p, \delta_1, p, \ldots, \delta_{m^*-1,1}, p, \delta_{m^*+1,1}, p, \ldots, \delta_{M,1}, p, \beta_{k+1,1}, p, \ldots, \beta_K, p)^T 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 \text{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})^T is a vector of corresponding values of the explanatory variables (including shares of observations in each interval of variable \text{xk} except for the reference interval \text{m^*}) or \text{allXVal } = (1, z_{1,1, \ldots, z_{1,1, T}, \ldots, z_{1,1, P}, \ldots, z_{T,1, P}) is a vector of the \( P \) sets of explanatory variables from the conditional logit (including shares of observations in each interval of variable \text{z1,p} except for the reference interval \text{m^*}); argument \text{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 \text{s1, \ldots, sM} of each
interval in the vectors \texttt{allCoef} and \texttt{allXVal}, whereas the position of the reference interval \( m^* \) is set to zero; argument \texttt{allCoefSE} is a vector of the standard errors of all coefficients in \texttt{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 \texttt{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 \texttt{yCat}, only in combination with option "MNL" or "CondL", indicates the \( p \)th output category of interest; and argument \texttt{xBound = (b_0, ..., b_M)\top}\) indicates the boundaries of the intervals as in function \texttt{linElaInt}.

\begin{verbatim}
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
\end{verbatim}
3.3. Effects of continuous variables when they change between discrete intervals

As in section 3.1, we assume the following model

\[
Pr(y = 1 | x) = \frac{\exp(\beta_0 + \sum_{k=1}^{K} \beta_k x_k)}{1 + \exp(\beta_0 + \sum_{k=1}^{K} \beta_k x_k)},
\]

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

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

\[
\approx \frac{\exp(\beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j + \beta_k E[x_k | b_{m-1} < x_k \leq b_m])}{1 + \exp(\beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j + \beta_k E[x_k | b_{m-1} < x_k \leq b_m])}
- \frac{\exp(\beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j + \beta_k E[x_k | b_{l-1} < x_k \leq b_l])}{1 + \exp(\beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j + \beta_k E[x_k | b_{l-1} < x_k \leq b_l])}
\]

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

For model specifications that additionally include a quadratic term of the explanatory variable,
\[
E_{k,ml} = E[y|b_{m-1} < x_k \leq b_m] - E[y|b_{t-1} < x_k \leq b_t]
\]
\[
\approx \frac{\exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k+1\}} \beta_j x_j + \beta_k E[x_k|b_{m-1} < x_k \leq b_m] \right)}{1 + \exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k+1\}} \beta_j x_j + \beta_k E[x_k|b_{m-1} < x_k \leq b_m] \right)}
\]
\[
- \frac{\exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k+1\}} \beta_j x_j + \beta_k E[x_k|b_{t-1} < x_k \leq b_t] \right)}{1 + \exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k+1\}} \beta_j x_j + \beta_k E[x_k|b_{t-1} < x_k \leq b_t] \right)} \tag{187}
\]

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-nominal logit function equation (185) modifies to
\[
E_{k,ml,p} \approx \frac{\exp \left( \beta_{0,p} + \sum_{j \in \{1, \ldots, K\} \setminus \{k\}} \beta_{j,p} x_j + \beta_{k,p} E[x_k|b_{m-1} < x_k \leq b_m] \right)}{1 + \sum_{p \in \{1, \ldots, P\} \setminus \{p\}} \exp \left( \beta_{0,p} + \sum_{j \in \{1, \ldots, K\} \setminus \{k\}} \beta_{j,p} x_j + \beta_{k,p} E[x_k|b_{m-1} < x_k \leq b_m] \right)}
\]
\[
- \frac{\exp \left( \beta_{0,p} + \sum_{j \in \{1, \ldots, K\} \setminus \{k\}} \beta_{j,p} x_j + \beta_{k,p} E[x_k|b_{t-1} < x_k \leq b_t] \right)}{1 + \sum_{p \in \{1, \ldots, P\} \setminus \{p\}} \exp \left( \beta_{0,p} + \sum_{j \in \{1, \ldots, K\} \setminus \{k\}} \beta_{j,p} x_j + \beta_{k,p} E[x_k|b_{t-1} < x_k \leq b_t] \right)} \tag{188}
\]

and equation (187) modifies accordingly.

In the case of the conditional logit function equation (185) modifies to
\[
E_{k,ml} \approx \frac{\exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, T\} \setminus \{t\}} \gamma_j z_{j,p} + \gamma_k E[z_k|b_{m-1} < z_k \leq b_m] \right)}{\sum_{p \in C} \exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, T\} \setminus \{t\}} \gamma_j z_{j,p} + \gamma_k E[z_k|b_{m-1} < z_k \leq b_m] \right)}
\]
\[
- \frac{\exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, T\} \setminus \{t\}} \gamma_j z_{j,p} + \gamma_k E[z_k|b_{t-1} < z_k \leq b_t] \right)}{\sum_{p \in C} \exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, T\} \setminus \{t\}} \gamma_j z_{j,p} + \gamma_k E[z_k|b_{t-1} < z_k \leq b_t] \right)} \tag{189}
\]

and equation (187) modifies accordingly.
And in the case of the nested logit function equation (185) modifies to

$$E_{k,ml,p,o} = \frac{\exp(\beta_0/\lambda_o + \sum_{j \in \{1,\ldots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_o E[x_{k,p}|b_{m-1} < x_k \leq b_m])}{\sum_{p \in B_o} \exp(\beta_0/\lambda_o + \sum_{j \in \{1,\ldots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_o E[x_{k,p}|b_{m-1} < x_k \leq b_m])} \cdot \frac{\exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o V_o)}{\sum_{o=1}^{O} \exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o V_o)}$$

$$\cdot \frac{\exp(\beta_0/\lambda_o + \sum_{j \in \{1,\ldots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_o E[x_{k,p}|b_{l-1} < x_k \leq b_l])}{\sum_{p \in B_o} \exp(\beta_0/\lambda_o + \sum_{j \in \{1,\ldots,K\}\setminus k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_o E[x_{k,p}|b_{l-1} < x_k \leq b_l])} \cdot \frac{\exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o V_o)}{\sum_{o=1}^{O} \exp(\gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o V_o)} \tag{190}$$

where equation (141) modifies to

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

and

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

respectively.

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

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

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

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

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

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

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

where

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

57
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 multinomial logit the gradient vector, \( \frac{\partial E_{k,ml,p}}{\partial \beta} \), modifies to:

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

(198)

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

(199)

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

(200)

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

(201)

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

(202)

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

(203)

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

(204)

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

(205)
where

\[ \exp_{m,p}(\cdot) \equiv \exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k+1\}} \beta_j x_j + \beta_{k,p} x_k | b_{m-1} < x_k \leq b_m \right) + \beta_{k+1,p} x_k^2 | b_{m-1} < x_k \leq b_m \]

and

\[ \exp_{l,p}(\cdot) \equiv \exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, K\} \setminus \{k+1\}} \beta_j x_j + \beta_{k,p} x_k | b_{t-1} < x_k \leq b_t \right) + \beta_{k+1,p} x_k^2 | b_{t-1} < x_k \leq b_t \). \]

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

\[
\frac{\partial E_{t,ml,p}}{\partial \gamma_0} = \frac{\exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot) - \exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot)}{\left( \sum_{p \in C} \exp_{m,p}(\cdot) \right)^2} \\
\frac{\partial E_{t,ml,p}}{\partial \gamma_j} = \frac{\exp_{m,p}(\cdot) z_{j,p} \cdot \sum_{p \in C} \exp_{m,p}(\cdot) - \exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot) z_{j,p}}{\left( \sum_{p \in C} \exp_{m,p}(\cdot) \right)^2} \\
\frac{\partial E_{t,ml,p}}{\partial \gamma_l} = \frac{\exp_{m,p}(\cdot) z_{l,p} \cdot \sum_{p \in C} \exp_{m,p}(\cdot) - \exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot) z_{l,p}}{\left( \sum_{p \in C} \exp_{m,p}(\cdot) \right)^2} \\
\frac{\partial E_{t,ml,p}}{\partial \gamma_{l+1}} = \frac{\exp_{m,p}(\cdot) z_{l+1,p} \cdot \sum_{p \in C} \exp_{m,p}(\cdot) - \exp_{m,p}(\cdot) \cdot \sum_{p \in C} \exp_{m,p}(\cdot) z_{l+1,p}}{\left( \sum_{p \in C} \exp_{m,p}(\cdot) \right)^2} \quad \forall \gamma_j \neq t, t+1
\]

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

\[ \exp_{l,p}(\cdot) \equiv \exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, T\} \setminus \{t, t+1\}} \gamma_j z_{j,p} + \gamma_{t,p} E[z_t | b_{t-1} < z_t \leq b_t] + \gamma_{t+1,p} E[z_t^2 | b_{t-1} < z_t \leq b_t] \right). \]
Finally, in the case of the nested logit, the gradient vector, \( \frac{\partial E_{t,ml,p}}{\partial \beta} \), modifies to:

\[
\frac{\partial E_{t,ml,p}}{\partial \beta_0} = \left( \frac{1}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{m,p,o}(\cdot) \frac{1}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{m,p,o}(\cdot)} \right) \pi_{m,p} \pi_{m,o} + \left( \sum_{p \in B_o} 1 - \frac{\sum_{o=1}^{O} \left( \exp_{m,o}(\cdot) \sum_{p \in B_o} 1 \right)}{\sum_{o=1}^{O} \exp_{m,o}(\cdot)} \right) \pi_{m,p} \pi_{m,o} - \left( \frac{1}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{l,p,o}(\cdot) \frac{1}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{l,p,o}(\cdot)} \right) \pi_{l,p} \pi_{l,o} + \left( \sum_{p \in B_o} 1 - \frac{\sum_{o=1}^{O} \left( \exp_{l,o}(\cdot) \sum_{p \in B_o} 1 \right)}{\sum_{o=1}^{O} \exp_{l,o}(\cdot)} \right) \pi_{l,p} \pi_{l,o}
\]

(210)

\[
\frac{\partial E_{t,ml,p}}{\partial \beta_j} = \left( \frac{x_{p,j}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{m,p,o}(\cdot) \frac{x_{p,j}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{m,p,o}(\cdot)} \right) \pi_{m,p} \pi_{m,o} + \left( \sum_{p \in B_o} x_{p,j} - \frac{\sum_{o=1}^{O} \left( \exp_{m,o}(\cdot) \sum_{p \in B_o} x_{p,j} \right)}{\sum_{o=1}^{O} \exp_{m,o}(\cdot)} \right) \pi_{m,p} \pi_{m,o} - \left( \frac{x_{p,j}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{l,p,o}(\cdot) \frac{x_{p,j}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{l,p,o}(\cdot)} \right) \pi_{l,p} \pi_{l,o} + \left( \sum_{p \in B_o} x_{p,j} - \frac{\sum_{o=1}^{O} \left( \exp_{l,o}(\cdot) \sum_{p \in B_o} x_{p,j} \right)}{\sum_{o=1}^{O} \exp_{l,o}(\cdot)} \right) \pi_{l,p} \pi_{l,o}
\]

(211)

\[
\frac{\partial E_{t,ml,p}}{\partial \beta_k} = \left( \frac{x_{p,m}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{m,p,o}(\cdot) \frac{x_{p,m}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{m,p,o}(\cdot)} \right) \pi_{m,p} \pi_{m,o} + \left( \sum_{p \in B_o} x_{p,m} - \frac{\sum_{o=1}^{O} \left( \exp_{m,o}(\cdot) \sum_{p \in B_o} x_{p,m} \right)}{\sum_{o=1}^{O} \exp_{m,o}(\cdot)} \right) \pi_{m,p} \pi_{m,o} - \left( \frac{x_{p,l}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{l,p,o}(\cdot) \frac{x_{p,l}}{\lambda_o} \right)}{\sum_{p \in B_o} \exp_{l,p,o}(\cdot)} \right) \pi_{l,p} \pi_{l,o} + \left( \sum_{p \in B_o} x_{p,l} - \frac{\sum_{o=1}^{O} \left( \exp_{l,o}(\cdot) \sum_{p \in B_o} x_{p,l} \right)}{\sum_{o=1}^{O} \exp_{l,o}(\cdot)} \right) \pi_{l,p} \pi_{l,o}
\]

(212)
\[
\frac{\partial E_{t,m,l,p}}{\partial \beta_{k+1}} = \left( \frac{x_{p,m+1}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \text{exp}_{m,p,o} \left( \cdot \right) x_{p,m+1} \right)}{\sum_{p \in B_o} \text{exp}_{m,p,o} \left( \cdot \right)} \right) \pi_{m,p} \pi_{m,o} \\
+ \left( \frac{\sum_{p \in B_o} x_{p,m+1}}{\lambda_o} - \frac{\sum_{o=1}^{O} \left( \text{exp}_{m,o} \left( \cdot \right) \sum_{p \in B_o} x_{p,m+1} \right)}{\sum_{o=1}^{O} \text{exp}_{m,o} \left( \cdot \right)} \right) \pi_{l,p} \pi_{l,o} \\
- \left( \frac{x_{p,l+1}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \text{exp}_{l,p,o} \left( \cdot \right) x_{p,l+1} \right)}{\sum_{p \in B_o} \text{exp}_{l,p,o} \left( \cdot \right)} \right) \pi_{l,p} \pi_{l,o} \\
+ \left( \frac{\sum_{p \in B_o} x_{p,l+1}}{\lambda_o} - \frac{\sum_{o=1}^{O} \left( \text{exp}_{l,o} \left( \cdot \right) \sum_{p \in B_o} x_{p,l+1} \right)}{\sum_{o=1}^{O} \text{exp}_{l,o} \left( \cdot \right)} \right) \pi_{l,p} \pi_{l,o} \\
\right)
\]

with

\[
\text{exp}_{m,p,o} \left( \cdot \right) \equiv \exp \left( \beta_0/\lambda_o + \sum_{j \in \{1, ..., K\} \backslash k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_o E[x_{k,p} | b_{m-1} < x_{k,p} \leq b_m] \right)
\]

\[
\text{exp}_{l,p,o} \left( \cdot \right) \equiv \exp \left( \beta_0/\lambda_o + \sum_{j \in \{1, ..., K\} \backslash k} \beta_j/\lambda_o x_{j,p} + \beta_k/\lambda_o E[x_{k,p} | b_{m-1} < x_{k,p} \leq b_m] \right)
\]

\[
\text{exp}_{m/l,o} \left( \cdot \right) \equiv \exp \left( \gamma_0 + \sum_{t=1}^{T} \gamma_t z_{t,o} + \lambda_o IV_o \right)
\]

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:

```r
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" )
    } else if( length( allCoefSE ) != nCoef ){
      # calculate effect
      # ...
    }
  } else if( method == "multi-nomial" ){
    # ...
  } else if( method == "conditional" ){
    # ...
  } else {
    stop( "method is not supported" )
  }
}
```
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" )
    }

    O <- 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" )
    }
}
```r
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:O ){
  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:O ){
    intXbeta[[ l ]] <- colSums( mCoef[ ,1 ] * mXValint[[ l ]] )
    refXbeta[[ l ]] <- colSums( mCoef[ ,1 ] * 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:O ){
    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(\text{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("MNL"
    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]*
    # #
\[
\begin{align*}
\text{sum( } & \exp( \text{intXbeta } yCat) \text{) } \times \text{rowSums( } \exp( \text{intXbeta } ) \times mXVal[ -xPos, ] \text{) } \)/
\left( \text{sum( } \exp( \text{intXbeta } ) \text{) } \right)^2 - \\
\exp( \text{refXbeta } yCat) \times mXVal[ -xPos, yCat] \times \\
\text{rowSums( } \exp( \text{refXbeta } ) \times mXVal[ -xPos, ] \text{) } \)/
\left( \text{sum( } \exp( \text{refXbeta } ) \text{) } \right)^2 - \
\exp( \text{refXbeta } yCat) \times \text{rowSums( } \exp( \text{refXbeta } ) \times mXVal[ -xPos, ] \text{) } \)/
\left( \text{sum( } \exp( \text{refXbeta } ) \text{) } \right)^2 - \
\exp( \text{refXbeta } yCat) \times \text{rowSums( } \exp( \text{refXbeta } ) \times mXVal[ -xPos, yCat] \text{) } \)/
\left( \text{sum( } \exp( \text{refXbeta } ) \text{) } \right)^2 - \
\end{align*}
\]
(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 = (β₀,...,βₖ)ᵀ a vector of all coefficients from a binary logit model, allCoef = (γ₀,...,γₜ)ᵀ a vector of all coefficients from a conditional logit model, allCoefBra = (β₀,...,βₖ)ᵀ, 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 = (β₀₁,...,βₖ₁,...,β₀ₚ,...,βₖₚ) 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,...,xₖ)ᵀ, a vector of all corresponding sample means and 1 for the intercept for the binary or multinomial logit, allXVal = (1,...,zₜ,1,...,1,...,zₜ,p) a vector of the P sets of explanatory variables from the conditional logit, or allXVal = (((1,...,xₖ,p,o),(1,...,xₖ,p,o)),...,((1,...,xₖ,p,o),...,(1,...,xₖ,p,o)))ᵀ, 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 = (se(β₀),...se(βₖ))ᵀ, 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 multinomial logit regression;

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

- argument refBound = (bₘ₋₁,bₘ) indicates the boundaries of the reference interval;

⁹The 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 \( \text{intBound} = (b_{m-1}, b_m) \) indicates the boundaries of the interval of interest;
- argument \( \text{method} = \text{c("binary", "MNL", "CondL", "NestL")} \) 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 \( \lambda \) 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 \text{allCoef} are already corrected by \( \frac{\beta_k}{\lambda_0} \);
- finally, \( y\text{Cat} \) a scalar identifying which of the \( P \) output categories is of interest, only in combination with \( \text{method} = \text{"MNL", "CondL", and "NestL"} \) (for the twig level), and \( y\text{CatBra} \) at the branch level.

```r
# 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, 7.5969066666667
eff7a
```
## effect stdEr
## 0.0004999485 NA

```r
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, 7.674666666666767
```

```r
eff7b
```

## effect stdEr
## 0.0004625630 0.0003872931

##Example
```r
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"
)
```

```r
eff8a
```

## effect stdEr
## -0.02698526 NA

```r
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"
)
```

```r
eff8b
```

## effect stdEr
## -0.02698526 0.01805266

##Example
```r
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"
)
```

```r
eff9a
```

## effect stdEr
## -0.02698526 0.01805266
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

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
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

3.4. Grouping and re-basing categorical variables

We consider a regression model

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

\[
D_m = \begin{cases} 
1 & \text{if } x_k \in c_m \\
0 & \text{otherwise}
\end{cases} \quad \forall \, m = 1,\ldots,M.
\]  \hspace{1cm} (218)

Like in section 1.4, the \( k \)th explanatory 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

\[
E_{k,lr} = E[y|x_k \in c_l^r] - E[y|x_k \in c_l^*] \hspace{1cm} (219)
\]

\[
= \frac{\exp\left(\beta_0 + \sum_{j \in \{1,...,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m E[D_m|x_k \in c_l^r]\right)}{1 + \exp\left(\beta_0 + \sum_{j \in \{1,...,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m E[D_m|x_k \in c_l^*]\right)}
\]  \hspace{1cm} (220)

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

\[
= \frac{\exp\left(\beta_0 + \sum_{j \in \{1,...,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{nl}\right)}{1 + \exp\left(\beta_0 + \sum_{j \in \{1,...,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{nl}\right)}
\]  \hspace{1cm} (221)

\[
- \frac{\exp\left(\beta_0 + \sum_{j \in \{1,...,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{nr}\right)}{1 + \exp\left(\beta_0 + \sum_{j \in \{1,...,K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{nr}\right)}
\]
where \( D_{mn} \) is defined as in equation (56). Whilst for the case of the multi-nominal logit regression, equation (52) modifies to

\[
E_{k,lr,p} = \frac{\exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\}} \beta_j p x_j + \sum_{m=1}^{M} \delta_{m,p} D_{ml} \right)}{1 + \sum_{p \in \{1, \ldots, P\}} \exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\}} \beta_j p x_j + \sum_{m=1}^{M} \delta_{m,p} D_{ml} \right)}
\]

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

\[
E_{t,lr,p} = \frac{\exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, T\}} \gamma_j p z_j + \sum_{m=1}^{M} \delta_{m,p} D_{ml} \right)}{\sum_{p \in C} \exp \left( \gamma_0 + \sum_{j \in \{1, \ldots, T\}} \gamma_j p z_j + \sum_{m=1}^{M} \delta_{m,p} D_{ml} \right)}
\]

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

\[
E_{t,lr,p,o} = \frac{\exp \left( \frac{\beta_0}{\lambda_o} + \sum_{j \in \{1, \ldots, K\}} \frac{\beta_j}{\lambda_o} x_j + \sum_{m=1}^{M} \delta_{m} / \lambda_o D_{ml} \right)}{\sum_{p \in B_o} \exp \left( \frac{\beta_0}{\lambda_o} + \sum_{j \in \{1, \ldots, K\}} \frac{\beta_j}{\lambda_o} x_j + \sum_{m=1}^{M} \delta_{m} / \lambda_o D_{ml} \right) \cdot \exp \left( \gamma_0 + \sum_{t=1}^{T} \gamma_t z_t + \lambda_o IV_{o,t,o} \right) \cdot \sum_{o=1}^{O} \exp \left( \frac{\beta_0}{\lambda_o} + \sum_{j \in \{1, \ldots, K\}} \frac{\beta_j}{\lambda_o} x_j + \sum_{m=1}^{M} \delta_{m} / \lambda_o D_{mr} \right) \cdot \exp \left( \gamma_0 + \sum_{t=1}^{T} \gamma_t z_t + \lambda_o IV_{o,r} \right) \cdot \sum_{o=1}^{O} \exp \left( \gamma_0 + \sum_{t=1}^{T} \gamma_t z_t + \lambda_o IV_{o,r} \right)}{\sum_{p \in B_o} \exp \left( \frac{\beta_0}{\lambda_o} + \sum_{j \in \{1, \ldots, K\}} \frac{\beta_j}{\lambda_o} x_j + \sum_{m=1}^{M} \gamma_m / \lambda_o D_{m,l/r} \right)}
\]

where

\[
IV_{o,l/r} = \ln \sum_{p \in B_o} \exp \left( \frac{\beta_0}{\lambda_o} + \sum_{j \in \{1, \ldots, K\}} \frac{\beta_j}{\lambda_o} x_j + \sum_{m=1}^{M} \gamma_m / \lambda_o D_{m,l/r} \right)
\]

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

\[
se \left( E_{k,lr} \right) = \sqrt{\left( \frac{\partial E_{k,lr}}{\partial (\beta)} \right)^\top \cdot \text{Var} \begin{pmatrix} \beta \\ \delta \end{pmatrix} \cdot \frac{\partial E_{k,lr}}{\partial (\beta)}},
\]

(226)
with the partial derivatives defined as

\[
\frac{\partial E_{k,l,r}}{\partial \beta_0} = \frac{\exp_{ml}(\cdot)}{[1 + \exp_{ml}(\cdot)]^2} - \frac{\exp_{mr}(\cdot)}{[1 + \exp_{mr}(\cdot)]^2} 
\]

(227)

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

(228)

\[
\frac{\partial E_{k,l,r}}{\partial \delta_m} = \frac{\exp_{ml}(\cdot)}{[1 + \exp_{ml}(\cdot)]^2} D_{ml} - \frac{\exp_{mr}(\cdot)}{[1 + \exp_{mr}(\cdot)]^2} D_{mr} \quad \forall m = 1, \ldots, M. 
\]

(229)

and for the multi-nomial logit as

\[
\frac{\partial E_{k,l,r,p}}{\partial \beta_0,p} = \frac{\exp_{ml,p}(\cdot) \cdot (1 + \sum_{o \in \{1, \ldots, P\} \setminus p,p^*} \exp_{ml,o}(\cdot))}{[1 + \sum_{p \in \{1, \ldots, P\} \setminus p^*} \exp_{ml,p}(\cdot)]^2} 
\]

(230)

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

(231)

\[
\frac{\partial E_{k,l,r,p}}{\partial \beta_{j,p}} = \frac{\exp_{ml,p}(\cdot) \cdot (1 + \sum_{o \in \{1, \ldots, P\} \setminus p,p^*} \exp_{ml,o}(\cdot)) \cdot x_j}{[1 + \sum_{p \in \{1, \ldots, P\} \setminus p^*} \exp_{ml,p}(\cdot)]^2} 
\]

(232)

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

(233)

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

(234)

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

(235)
with
\[
\exp_{ml}(\cdot) \equiv \exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{ml} \right) \quad (236)
\]
\[
\exp_{mr}(\cdot) \equiv \exp \left( \beta_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j x_j + \sum_{m=1}^{M} \delta_m D_{mr} \right) \quad (237)
\]

and for the conditional logit as
\[
\frac{\partial E_{t,lr}}{\partial \gamma_0} = \frac{\exp_{ml,p}(\cdot) \cdot \sum_{p \in C} \exp_{ml,p} - \exp_{ml,p}(\cdot) \cdot \sum_{p \in C} \exp_{ml,p}}{[\sum_{p \in C} \exp_{ml}(\cdot)]^2}
\]
\[
- \frac{\exp_{mr,p}(\cdot) \cdot \sum_{p \in C} \exp_{mr,p} - \exp_{mr,p}(\cdot) \cdot \sum_{p \in C} \exp_{mr,p}}{[\sum_{p \in C} \exp_{mr}(\cdot)]^2}
\]

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

\[
\frac{\partial E_{t,lr}}{\partial \delta_m} = \frac{\exp_{ml,p}(\cdot) D_{ml} \cdot \sum_{p \in C} \exp_{ml,p} - \exp_{ml,p}(\cdot) \cdot \sum_{p \in C} \exp_{ml,p} D_{ml}}{[\sum_{p \in C} \exp_{ml}(\cdot)]^2}
\]
\[
- \frac{\exp_{mr,p}(\cdot) D_{mr} \cdot \sum_{p \in C} \exp_{mr,p} - \exp_{mr,p}(\cdot) \cdot \sum_{p \in C} \exp_{mr,p} D_{mr}}{[\sum_{p \in C} \exp_{mr}(\cdot)]^2} \quad \forall \ m = 1, \ldots, M
\]
and for the nested logit as

$$\frac{\partial E_{o,p,l}}{\partial \beta_0} = \left( \frac{1}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{o,p,l} \left( \frac{1}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,l} \left( \cdot \right)} \right) + \left( \sum_{p \in B_o} \frac{\left( \sum_{p \in B_o} \left( \exp_{o,p,l} \left( \frac{1}{\lambda_o} \right) \right) \right)}{\sum_{p \in B_o} \exp_{o,p,l} \left( \cdot \right)} \right) \cdot \pi_{p,l} \pi_{o,l}$$

$$- \left( \left( \frac{1}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{o,p,r} \left( \frac{1}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,r} \left( \cdot \right)} \right) \right) + \left( \sum_{p \in B_o} \frac{\left( \sum_{p \in B_o} \left( \exp_{o,p,r} \left( \frac{1}{\lambda_o} \right) \right) \right)}{\sum_{p \in B_o} \exp_{o,p,r} \left( \cdot \right)} \right) \cdot \pi_{p,r} \pi_{o,r}$$

$$= \left( \frac{x_{j,p}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{o,p,l} \left( \frac{x_{j,p}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,l} \left( \cdot \right)} \right) + \left( \frac{\sum_{p \in B_o} \left( \exp_{o,p,l} \left( \frac{x_{j,p}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,l} \left( \cdot \right)} \right) \cdot \pi_{p,l} \pi_{o,l}$$

$$- \left( \frac{x_{j,p}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{o,p,r} \left( \frac{x_{j,p}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,r} \left( \cdot \right)} \right) + \left( \sum_{p \in B_o} \frac{\left( \sum_{p \in B_o} \left( \exp_{o,p,r} \left( \frac{x_{j,p}}{\lambda_o} \right) \right) \right)}{\sum_{p \in B_o} \exp_{o,p,r} \left( \cdot \right)} \right) \cdot \pi_{p,r} \pi_{o,r}$$

$$= \left( \frac{D_{m,p,l}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{o,p,l} \left( \frac{D_{m,p,l}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,l} \left( \cdot \right)} \right) + \left( \frac{\sum_{p \in B_o} \left( \exp_{o,p,l} \left( \frac{D_{m,p,l}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,l} \left( \cdot \right)} \right) \cdot \pi_{p,l} \pi_{o,l}$$

$$- \left( \frac{x_{j,p}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{o,p,r} \left( \frac{D_{m,p,r}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,r} \left( \cdot \right)} \right) + \left( \sum_{p \in B_o} \frac{\left( \sum_{p \in B_o} \left( \exp_{o,p,r} \left( \frac{D_{m,p,r}}{\lambda_o} \right) \right) \right)}{\sum_{p \in B_o} \exp_{o,p,r} \left( \cdot \right)} \right) \cdot \pi_{p,r} \pi_{o,r}$$

$$= \left( \frac{D_{m,p,l}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{o,p,l} \left( \frac{D_{m,p,l}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,l} \left( \cdot \right)} \right) + \left( \frac{\sum_{p \in B_o} \left( \exp_{o,p,l} \left( \frac{D_{m,p,l}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,l} \left( \cdot \right)} \right) \cdot \pi_{p,l} \pi_{o,l}$$

$$- \left( \frac{x_{j,p}}{\lambda_o} - \frac{\sum_{p \in B_o} \left( \exp_{o,p,r} \left( \frac{x_{j,p}}{\lambda_o} \right) \right)}{\sum_{p \in B_o} \exp_{o,p,r} \left( \cdot \right)} \right) + \left( \sum_{p \in B_o} \frac{\left( \sum_{p \in B_o} \left( \exp_{o,p,r} \left( \frac{x_{j,p}}{\lambda_o} \right) \right) \right)}{\sum_{p \in B_o} \exp_{o,p,r} \left( \cdot \right)} \right) \cdot \pi_{p,r} \pi_{o,r}$$
The following function calculates the effect \( E_{k,lr} \) and its standard error in accordance with equations (221), (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):

```r
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" )
      }
    }
  }
}
```

with

\[
\exp_{p,o,l}(\cdot) \equiv \exp \left( \beta_0/\lambda_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j/\lambda_0 x_{j,p} + \sum_{m=1}^M \delta_m/\lambda_0 D_{m,p,l} \right)
\] (245)

\[
\exp_{p,o,r}(\cdot) \equiv \exp \left( \beta_0/\lambda_0 + \sum_{j \in \{1, \ldots, K\} \setminus k} \beta_j/\lambda_0 x_{j,p} + \sum_{m=1}^M \delta_m/\lambda_0 D_{m,p,l} \right)
\] (246)

\[
\exp_{t,r,o}(\cdot) \equiv \exp \left( \gamma_0 + \sum_{t=1}^T \gamma_t z_{t,o} + \lambda_0 IV_o \right)
\] (247)
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
  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
  # effeG <- exp( XBetaEffect )/( 1 + exp( XBetaEffect ) ) - exp( XBetaRef )/( 1 + exp( XBetaEffect ) )
} else if( method == "MNL" ){
  # D_mr
  XBetaRef <- colSums( mCoef[ -xPos, , drop = FALSE ] * x
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 ]
    }
  }

79
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 ]
}

dervCoef <- 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, yCat ] ) )/
( sum( exp( XBetaEffect ) ) )^2 -
( exp( XBetaRef[ yCat ] ) * mXVal[ -xPos, yCat ] *
sum( exp( XBetaRef ) ) -
exp( XBetaRef[ yCat ] ) * sum( exp( XBetaRef ) *
mXVal[ -xPos, yCat ] ) )/
( 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)
v covCoef <- diag( allCoefSE^2 )
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

\[
\text{allCoef} = (\beta_0, \ldots, \beta_{k-1}, \beta_{k+1}, \ldots, \beta_K, \delta_1, \ldots, \delta_M, 1, \beta_0, \ldots)
\]

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 \(k\)th 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 \(k\)th variable which includes the share of the reference group of the \(k\)th variable or

\[
\text{allXVal} = (z_{0,1}, \ldots, z_{t-1,1}, z_{t+1,1}, \ldots, z_{T,1}, s_{1,1}, \ldots, s_{M,1}, \ldots, z_{0,P}, \ldots, z_{t-1,P}, z_{t+1,P}, \ldots, z_{T,P}, s_{1,P}, \ldots, s_{M,P})^\top
\]

are the mean values of the \(P\) sets of explanatory variables and the \(P\) sets of shares of the \(t\)th variable which includes the share of the reference group of the \(t\)th variable; argument `xPos` is a vector of at least two elements indicating the position of the \(k\)th or \(t\)th 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

\[
\text{allCoefSE} = (\text{se}(\beta_0), \ldots, \text{se}(\beta_{k-1}), \text{se}(\beta_{k+1}), \ldots, \text{se}(\beta_K), \text{se}(\delta_1), \ldots, \text{se}(\delta_M), 0, \ldots, 0, \ldots, \text{se}(\beta_{k-1,1}), \ldots, \text{se}(\beta_{k+1,1}), \ldots, \text{se}(\beta_{k+1,P}), \ldots, \text{se}(\beta_{k+1,P}))^\top
\]

are the standard errors of all coefficients of the multi-nomial logit regression which includes the share of the reference group of the \(k\)th variable, 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 \(p\)th 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 omission must be consistent for all four arguments.

### Example

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

```r
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
```

```
# 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
```
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

82
APPENDIX

A. Additional helper functions

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

```r
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:

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

83