Package 'qgcomp'

May 9, 2026

Title Quantile G-Computation

Version 2.18.10

Date 2026-03-24

Description G-computation for a set of time-fixed exposures with quantile-based basis functions, possibly under linearity and homogeneity assumptions. This approach estimates a regression line corresponding to the expected change in the outcome (on the link basis) given a simultaneous increase in the quantile-based category for all exposures. Works with continuous, binary, and right-censored time-to-event outcomes. Reference: Alexander P. Keil, Jessie P. Buckley, Katie M. OBrien, Kelly K. Ferguson, Shanshan Zhao, and Alexandra J. White (2019) A quantile-based g-computation approach to addressing the effects of exposure mixtures; <doi:10.1289/EHP5838>.

License GPL (>= 2)

URL https://github.com/alexpkeil1/qgcomp/

BugReports https://github.com/alexpkeil1/qgcomp/issues

Depends R (>= 3.5.0)

Imports AER, arm, future, future.apply, generics, ggplot2 (>= 3.3.0), grDevices, grid, gridExtra, nnet, numDeriv, pscl, rootSolve, stats, survival, tibble

Suggests broom, devtools, knitr, markdown, MASS, mice

VignetteBuilder knitr

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Alexander Keil [aut, cre]

Maintainer Alexander Keil <alex.keil@nih.gov>

Repository CRAN

Date/Publication 2026-03-24 13:20:02 UTC

Contents

`.qgcomp_object` *Creating a* `qgcompfit` *object*

Description

`.qgcomp_object` developer function to create a `qgcompfit` object (a list of class "qgcompfit")

Usage

```
.qgcomp_object(...)
```

Arguments

`...`	named objects to add to the `qgcompfit` object

Details

This is not a generally useful function, except for developers, who need to add items to an existing qgcompfit object.

Value

a `qgcompfit` object

Examples

```
set.seed(50)
# linear model, adding an arbitrary string to the object
dat <- data.frame(y=runif(50,-1,1), x1=runif(50), x2=runif(50), z=runif(50))
expnms = c('x1')
q=NULL
ft = glm(f=y ~ z + x1 + x2, data=dat, family=gaussian())
z = coef(ft)[2]/sqrt(vcov(ft)[2,2])
pval = (1-pnorm(abs(z)))*2
.qgcomp_object(fit = ft, coef=coef(ft)[2], q=NULL, var.coef = vcov(ft)[2,2],
ci.coef=coef(ft)[2] + c(-1.96, 1.96)*sqrt(vcov(ft)[2,2]), tstat = z, pval = pval,
bootstrap=FALSE)
```

.qgcomp_object_add *Adding objects to a* qgcompfit *object*

Description

.qgcomp_object_add developer function to add items to an existing qgcompfit object

Usage

```
.qgcomp_object_add(x, ..., overwrite_duplicates = TRUE)
```

Arguments

x	qgcompfit object
...	named objects to add to the qgcompfit object
overwrite_duplicates	(logical) overwrite list items in qgcompfit object with arguments that have matching names in ...

Details

This is not a generally useful function, except for developers, who need to add items to an existing qgcompfit object.

Value

a qgcompfit object

Examples

```
set.seed(50)
# linear model, adding an arbitrary string to the object
dat <- data.frame(y=runif(50,-1,1), x1=runif(50), x2=runif(50), z=runif(50))
ft = qgcomp.glm.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=gaussian())
ft2 = .qgcomp_object_add(ft, date=Sys.Date())
ft2$date # not typically part of a qgcomp object, but could be useful for e.g. datestamping analyses
```

checknames *Check for valid model terms in a qgcomp fit*

Description

This is an internal function called by qgcomp, qgcomp.glm.boot, and qgcomp.glm.noboot, but is documented here for clarity. Generally, users will not need to call this function directly. This function tries to determine whether there are non-linear terms in the underlying model, which helps infer whether the appropriate function is called, and whether more explicit function calls are needed.

Usage

```
checknames(terms)
```

Arguments

terms	model terms from attr(terms(modelfunction, data), "term.labels")

coxmsm_fit *Marginal structural Cox model (MSM) fitting within quantile g-computation*

Description

this is an internal function called by qgcomp.cox.noboot, qgcomp.cox.boot, and qgcomp.cox.noboot, but is documented here for clarity. Generally, users will not need to call this function directly.

Usage

```
coxmsm_fit(
  f,
  qdata,
  intvals,
  expnms,
  main = TRUE,
  degree = 1,
  id = NULL,
  weights,
  cluster = NULL,
  MCsize = 10000,
  ...
)
```

Arguments

f	an R formula representing the conditional model for the outcome, given all exposures and covariates. Interaction terms that include exposure variables should be represented via the AsIs function. Offset terms can be included via Surv(time,event) ~ exposure + offset(z)
qdata	a data frame with quantized exposures (as well as outcome and other covariates)
intvals	sequence, the sequence of integer values that the joint exposure is 'set' to for estimating the msm. For quantile g-computation, this is just 0:(q-1), where q is the number of quantiles of exposure.
expnms	a character vector with the names of the columns in qdata that represent the exposures of interest (main terms only!)
main	logical, internal use: produce estimates of exposure effect (psi) and expected outcomes under g-computation and the MSM
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic. Default=1)
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	"case weights" - passed to the "weight" argument of coxph
cluster	not yet implemented
MCsize	integer: sample size for simulation to approximate marginal hazards ratios
...	arguments to coxph (e.g. ties)

Details

This function first computes expected outcomes under hypothetical interventions to simultaneously set all exposures to a specific quantile. These predictions are based on g-computation, where the exposures are 'quantized', meaning that they take on ordered integer values according to their ranks, and the integer values are determined by the number of quantile cutpoints used. The function then takes these expected outcomes and fits an additional model (a marginal structural model) with the expected outcomes as the outcome and the intervention value of the exposures (the quantile integer) as the exposure. Under causal identification assumptions and correct model specification, the MSM yields a causal exposure-response representing the incremental change in the expected outcome given a joint intervention on all exposures.

See Also

qgcomp.cox.boot, and qgcomp.cox.noboot

Examples

```
set.seed(50)
dat <- data.frame(time=(tmg <- pmin(.1,rweibull(50, 10, 0.1))), d=1.0*(tmg<0.1),
                  x1=runif(50), x2=runif(50), z=runif(50))
expnms=paste0("x", 1:2)
qdata  = quantize(dat, expnms)$data
```

```
f = survival::Surv(time, d)~x1 + x2
fit <- survival::coxph(f, data = qdata, y=TRUE, x=TRUE)
r1 = qdata[1,,drop=FALSE]
times = survival::survfit(fit, newdata=r1, se.fit=FALSE)$time
(obj <- coxmsm_fit(f, qdata, intvals=c(0,1,2,3), expnms, main=TRUE, degree=1,
   id=NULL, MCsize=100))
#dat2 <- data.frame(psi=seq(1,4, by=0.1))
#summary(predict(obj))
#summary(predict(obj, newdata=dat2))
```

`glance.qgcompfit` *Glance at a qgcompfit object*

Description

Glance accepts a model object and returns a tibble::tibble() with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA. (Description taken from `broom::glance` help file.)

Usage

```
## S3 method for class 'qgcompfit'
glance(x, ...)
```

Arguments

`x`	a qgcompfit object
`...`	Not used

`homogeneity_test` *Hypothesis testing about a joint effect of exposures on a multinomial outcome*

Description

Tests the null hypothesis that the joint effect of the mixture components is homogenous across all referent outcome types

Usage

```
homogeneity_test(x, ...)

## S3 method for class 'qgcompmultfit'
homogeneity_test(x, ...)
```

Arguments

x	Result from qgcomp multinomial fit (qgcompmultfit object).
...	Unused

`hurdlemsm_fit` *Secondary prediction method for the (hurdle) qgcomp MSM.*

Description

this is an internal function called by `qgcomp.hurdle.boot`, but is documented here for clarity. Generally, users will not need to call this function directly.

Usage

```
hurdlemsm_fit(
  f,
  qdata,
  intvals,
  expnms,
  main = TRUE,
  degree = 1,
  id = NULL,
  weights,
  MCsize = 10000,
  containmix = list(count = TRUE, zero = TRUE),
  bayes = FALSE,
  x = FALSE,
  msmcontrol = hurdlemsm_fit.control(),
  ...
)
```

Arguments

f	an r formula representing the conditional model for the outcome, given all exposures and covariates. Interaction terms that include exposure variables should be represented via the `AsIs` function
qdata	a data frame with quantized exposures (as well as outcome and other covariates)
intvals	sequence, the sequence of integer values that the joint exposure is 'set' to for estimating the msm. For quantile g-computation, this is just 0:(q-1), where q is the number of quantiles of exposure.

expnms	a character vector with the names of the columns in qdata that represent the exposures of interest (main terms only!)
main	logical, internal use: produce estimates of exposure effect (psi) and expected outcomes under g-computation and the MSM
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic. Default=1)
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	not yet implemented
MCsize	integer: sample size for simulation to approximate marginal hazards ratios
containmix	named list of logical scalars with names "count" and "zero"
bayes	not used
x	keep design matrix? (logical)
msmcontrol	named list from hurdlemsm_fit.control
...	arguments to hurdle (e.g. dist)

Details

This function first computes expected outcomes under hypothetical interventions to simultaneously set all exposures to a specific quantile. These predictions are based on g-computation, where the exposures are 'quantized', meaning that they take on ordered integer values according to their ranks, and the integer values are determined by the number of quantile cutpoints used. The function then takes these expected outcomes and fits an additional model (a marginal structural model) with the expected outcomes as the outcome and the intervention value of the exposures (the quantile integer) as the exposure. Under causal identification assumptions and correct model specification, the MSM yields a causal exposure-response representing the incremental change in the expected outcome given a joint intervention on all exposures.

See Also

qgcomp.cox.boot, and qgcomp.cox.noboot

Examples

```
set.seed(50)
n=100
## Not run:
dat <- data.frame(y=rbinom(n, 1, 0.5)*rpois(n, 1.2), x1=runif(n), x2=runif(n), z=runif(n))
expnms = c("x1", "x2")
q = 4
qdata = quantize(dat, q=q, expnms=expnms)$data
f = y ~ x1 + x2 + z | 1
msmfit <- hurdlemsm_fit(f, qdata, intvals=(1:q)-1, expnms, main=TRUE,
  degree=1, id=NULL, MCsize=10000, containmix=list(count=TRUE, zero=FALSE),
  x=FALSE)
msmfit$msmfit
```

```
## End(Not run)
```

`hurdlemsm_fit.control` *Control of fitting parameters for zero inflated MSMs*

Description

this is an internal function called by `qgcomp.hurdle.boot`, but is documented here for clarity. Generally, users will not need to call this function directly.

Usage

```
hurdlemsm_fit.control(predmethod = rev(c("components", "catprobs")))
```

Arguments

`predmethod`	character in c("components", "catprobs"). "components" simulates from the model parameters directly while "catprobs" simulates outcomes from the category specific probabilities, which is output from predict.hurdle. The former is slightly more flexible and stable, but the latter is preferred in zero inflated negative bionomial models.

Details

Provides fine control over zero inflated MSM fitting

`joint_test` *Hypothesis testing about a joint effect of exposures on a multinomial outcome*

Description

Tests the null hypothesis that the joint effect of the mixture components is null across all referent outcome types (Test of global null effect of the mixture on a quantized basis)

Usage

```
joint_test(x, ...)

## S3 method for class 'qgcompmultfit'
joint_test(x, ...)
```

Arguments

`x`	Result from qgcomp multinomial fit (qgcompmultfit object).
`...`	Unused

Value

qgcompmulttest object (list) with results of a chi-squared test

---

`metals` *Well water data*

Description

Simulated well water measurements in North Carolina: 16 metals, 6 water chemistry measures, and 2 health outcomes (y = continuous; disease_state = binary/time-to-event in combination with disease_time)

A dataset containing well water measurements and health outcomes for 253 individuals. All continuous variables are standardized to have mean 0, standard deviation 1.

Usage

```
data(metals)
```

Format

A data frame with 253 rows and 24 variables:

y continuous birth outcome

disease_state binary outcome

disease_time time-to-disease_state: survival outcome censored at approximately the median

arsenic metal

barium metal

cadmium metal

calcium metal

chloride metal

chromium metal

copper metal

iron metal

lead metal

magnesium metal

manganese metal

mercury metal

selenium metal

silver metal

sodium metal

zinc metal

mage35 Binary covariate: maternal age > 35

nitrate water chemistry measure

nitrite water chemistry measure

sulfate water chemistry measure

ph water chemistry measure

total_alkalinity water chemistry measure

total_hardness water chemistry measure

`mice.impute.leftcenslognorm`

Imputation for limits of detection problems

Description

This function integrates with `mice` to impute values below the LOD using a left censored log-normal distribution. Note that "tobit" is an alias that uses a familiar term for this model.

Usage

```
mice.impute.leftcenslognorm(
  y,
  ry,
  x,
  wy = NULL,
  lod = NULL,
  debug = FALSE,
  ...
)

mice.impute.tobit(y, ry, x, wy = NULL, lod = NULL, debug = FALSE, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length length(y) indicating the the subset of elements in y to which the imputation model is fitted. The ry generally distinguishes the observed (TRUE) and missing values (FALSE) in y.
`x`	Numeric design matrix with length(y) rows with predictors for y. Matrix x may have no missing values.
`wy`	Logical vector of length length(y). A TRUE value indicates locations in y for which imputations are created.
`lod`	numeric vector of limits of detection (must correspond to index in original data) OR list in which each element corresponds to observation level limits of detection for each variable (list index must correspond to index in original data)
`debug`	logical, print extra info
`...`	arguments to `survreg`

Details

While this function has utility far beyond qgcomp, it is included in the qgcomp package because it will be useful for a variety of settings in which qgcomp is useful. Note that LOD problems where the LOD is small, and the q parameter from qgcomp.glm.noboot or qgcomp.glm.boot is not large, the LOD may be below the lowest quantile cutpoint which will yield identical datasets from the MICE procedure in terms of quantized exposure data. If only exposures are missing, and they have low LODs, then there will be no benefit in qgcomp from using MICE rather than imputing some small value below the LOD.

Value

Vector with imputed data, same type as y, and of length sum(wy)

Examples

```
N = 100
set.seed(123)
dat <- data.frame(y=runif(N), x1=runif(N), x2=runif(N), z=runif(N))
true = qgcomp.glm.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'),
        data=dat, q=2, family=gaussian())
mdat <- dat
mdat$x1 = ifelse(mdat$x1>0.5, mdat$x1, NA)
mdat$x2 = ifelse(mdat$x2>0.75, mdat$x2, NA)
cc <- qgcomp.glm.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'),
       data=mdat[complete.cases(mdat),], q=2, family=gaussian())

## Not run:
# note the following example imputes from the wrong parametric model and is expected to be biased
# as a result (but it demonstrates how to use qgcomp and mice together)
library("mice")
library("survival")
set.seed(1231)
impdat = mice(data = mdat,
  method = c("", "leftcenslognorm", "leftcenslognorm", ""),
  lod=c(NA, 0.5, 0.75, NA), debug=FALSE, m=10)
qc.fit.imp <- mira(
  call = call("qgcomp.glm.noboot(y~., expnms = c('x1', 'x2'), family=gaussian())"),
  call1 = impdat$call,
  nmis = impdat$nmis,
  analyses = lapply(1:10, function(x) qgcomp.glm.noboot(y~., expnms = c("x1", "x2"),
    data=complete(impdat, x), family=gaussian(), bayes=TRUE))
 )
#alternative way to specify limits of detection (useful if not all observations have same limit)
lodlist = list(rep(NA, N), rep(0.5, N), rep(0.75, N), rep(NA, N))
#lodlist = data.frame(rep(NA, N), rep(0.5, N), rep(0.75, N), rep(NA, N)) # also works
set.seed(1231)
impdat_alt = mice(data = mdat,
  method = c("", "leftcenslognorm", "leftcenslognorm", ""),
  lod=lodlist, debug=FALSE, m=10)
qc.fit.imp_alt <- mira(
  call = call("qgcomp.glm.noboot(y~., expnms = c('x1', 'x2'), family=gaussian())"),
```

```
  call1 = impdat_alt$call,
  nmis = impdat_alt$nmis,
  analyses = lapply(1:10, function(x) qgcomp.glm.noboot(y~., expnms = c("x1", "x2"),
    data=complete(impdat_alt, x), family=gaussian(), bayes=TRUE))
)
obj = pool(qc.fit.imp)
obj_alt = pool(qc.fit.imp_alt)
# true values
true
# complete case analysis
cc
# MI based analysis (identical answers for different ways to specify limits of detection)
summary(obj)
summary(obj_alt)

# summarizing weights (note that the weights should *not* be pooled
#    because they mean different things depending on their direction)
expnms = c("x1", "x2")
wts = as.data.frame(t(sapply(qc.fit.imp$analyses,
      function(x) c(-x$neg.weights, x$pos.weights)[expnms])))
eachwt = do.call(c, wts)
expwts = data.frame(Exposure = rep(expnms, each=nrow(wts)), Weight=eachwt)
library(ggplot2)
ggplot(data=expwts)+ theme_classic() +
  geom_point(aes(x=Exposure, y=Weight)) +
  geom_hline(aes(yintercept=0))

# this function can be used to impute from an intercept only model.
# but you need to "trick" mice to bypass checks for collinearity by including
# a variable that does not need to have values imputed (here, y).
# The internal collinearity checks by the mice package remove collinear variables
# and then throws an error if no predictor variabls are retained. Here, the
# trick is to use the "predictorMatrix" parameter to "impute" the non-missing
# variable y using x1 (which does nothing), and remove all predictors from the model for x1.
# This function only imputes x1 from a log normal distribution and thus is similar
# to many published examples of imputation from a Tobit model.

impdat2 = mice(data = mdat[,c("y","x1")],
  method = c("", "tobit"), remove.collinear=FALSE,
  lod=c(NA, 0.5), debug=FALSE, m=1,
  maxit=1, # maxit=1 because there is only 1 variable to impute
  predictorMatrix = as.matrix(rbind(c(0,1), c(0,0))))
plot(density(complete(impdat2, 1)$x1))

# now with survival data (very similar)
impdat = mice(data = mdat,
  method = c("", "tobit", "tobit", ""),
  lod=c(NA, 0.5, 0.75, NA), debug=FALSE)
qc.fit.imp <- mira(
  call = call("qgcomp.cox.noboot(Surv(y)~., expnms = c('x1', 'x2'))"),
  call1 = impdat$call,
  nmis = impdat$nmis,
  analyses = lapply(1:5, function(x) qgcomp.cox.noboot(Surv(y)~., expnms = c("x1", "x2"),
```

```
    data=complete(impdat, x)))
)
obj = pool(qc.fit.imp)
# MI based analysis
summary(obj)


## End(Not run)
```

`modelbound.boot` *Estimating qgcomp regression line confidence bounds*

Description

Calculates: expected outcome (on the link scale), and upper and lower confidence intervals (both pointwise and simultaneous)

Usage

```
modelbound.boot(x, alpha = 0.05, pwonly = FALSE)
```

Arguments

`x`	"qgcompfit" object from `qgcomp.glm.boot`,
`alpha`	alpha level for confidence intervals
`pwonly`	logical: return only pointwise estimates (suppress simultaneous estimates)

Details

This method leverages the bootstrap distribution of qgcomp model coefficients to estimate pointwise regression line confidence bounds. These are defined as the bounds that, for each value of the independent variable X (here, X is the joint exposure quantiles) the 95% bounds (for example) for the model estimate of the regression line E(Y|X) are expected to include the true value of E(Y|X) in 95% of studies. The "simultaneous" bounds are also calculated, and the 95% simultaneous bounds contain the true value of E(Y|X) for all values of X in 95% of studies. The latter are more conservative and account for the multiple testing implied by the former. Pointwise bounds are calculated via the standard error for the estimates of E(Y|X), while the simultaneous bounds are estimated using the bootstrap method of Cheng (reference below). All bounds are large sample bounds that assume normality and thus will be underconservative in small samples. These bounds may also inclue illogical values (e.g. values less than 0 for a dichotomous outcome) and should be interpreted cautiously in small samples.

Reference:

Cheng, Russell CH. "Bootstrapping simultaneous confidence bands." Proceedings of the Winter Simulation Conference, 2005.. IEEE, 2005.

Value

A data frame containing

linpred: The linear predictor from the marginal structural model

r/o/m: The canonical measure (risk/odds/mean) for the marginal structural model link

se....: the stndard error of linpred

ul..../ll....: Confidence bounds for the effect measure, and bounds centered at the canonical measure (for plotting purposes)

The confidence bounds are either "pointwise" (pw) and "simultaneous" (simul) confidence intervals at each each quantized value of all exposures.

See Also

qgcomp.glm.boot

Examples

```
set.seed(12)
## Not run:
dat <- data.frame(x1=(x1 <- runif(50)), x2=runif(50), x3=runif(50), z=runif(50),
                  y=runif(50)+x1+x1^2)
ft <- qgcomp.glm.boot(y ~ z + x1 + x2 + x3, expnms=c('x1','x2','x3'), data=dat, q=5)
modelbound.boot(ft, 0.05)

## End(Not run)
```

---

modelbound.ee *Estimating qgcomp regression line confidence bounds*

Description

Calculates: expected outcome (on the link scale), and upper and lower confidence intervals (both pointwise and simultaneous)

Usage

```
modelbound.ee(x, alpha = 0.05, pwonly = FALSE)
```

Arguments

x	"qgcompfit" object from qgcomp.glm.boot,
alpha	alpha level for confidence intervals
pwonly	logical: return only pointwise estimates (suppress simultaneous estimates)

Details

This method leverages the bootstrap distribution of qgcomp model coefficients to estimate pointwise regression line confidence bounds. These are defined as the bounds that, for each value of the independent variable X (here, X is the joint exposure quantiles) the 95% bounds (for example) for the model estimate of the regression line E(Y|X) are expected to include the true value of E(Y|X) in 95% of studies. The "simultaneous" bounds are also calculated, and the 95% simultaneous bounds contain the true value of E(Y|X) for all values of X in 95% of studies. The latter are more conservative and account for the multiple testing implied by the former. Pointwise bounds are calculated via the standard error for the estimates of E(Y|X), while the simultaneous bounds are estimated using the bootstrap method of Cheng (reference below). All bounds are large sample bounds that assume normality and thus will be underconservative in small samples. These bounds may also inclue illogical values (e.g. values less than 0 for a dichotomous outcome) and should be interpreted cautiously in small samples.

Reference:

Cheng, Russell CH. "Bootstrapping simultaneous confidence bands." Proceedings of the Winter Simulation Conference, 2005.. IEEE, 2005.

Value

A data frame containing

linpred: The linear predictor from the marginal structural model

r/o/m: The canonical measure (risk/odds/mean) for the marginal structural model link

se....: the stndard error of linpred

ul..../ll....: Confidence bounds for the effect measure, and bounds centered at the canonical measure (for plotting purposes)

The confidence bounds are either "pointwise" (pw) and "simultaneous" (simul) confidence intervals at each each quantized value of all exposures.

See Also

qgcomp.glm.boot

Examples

```
set.seed(12)
## Not run:
dat <- data.frame(x1=(x1 <- runif(50)), x2=runif(50), x3=runif(50), z=runif(50),
                  y=runif(50)+x1+x1^2)
ft <- qgcomp.glm.eet(y ~ z + x1 + x2 + x3, expnms=c('x1','x2','x3'), data=dat, q=5)
modelbound.ee(ft, 0.05)

## End(Not run)
```

msm.predict *Secondary prediction method for the (non-survival) qgcomp MSM.*

Description

this is an internal function called by qgcomp.glm.boot, but is documented here for clarity. Generally, users will not need to call this function directly.

Get predicted values from a qgcompfit object from qgcomp.glm.boot.

Usage

```
msm.predict(object, newdata = NULL)
```

Arguments

object	"qgcompfit" object from qgcomp.glm.boot function
newdata	(optional) new set of data (data frame) with a variable called psi representing the joint exposure level of all exposures under consideration

Details

(Not usually called by user) Makes predictions from the MSM (rather than the conditional g-computation fit) from a "qgcompfit" object. Generally, this should not be used in favor of the default predict.qgcompfit function. This function can only be used following the qgcomp.glm.boot function. For the qgcomp.glm.noboot function, predict.qgcompfit gives identical inference to predicting from an MSM.

Examples

```
set.seed(50)
dat <- data.frame(y=runif(50), x1=runif(50), x2=runif(50), z=runif(50))
obj <- qgcomp.glm.boot(y ~ z + x1 + x2 + I(z*x1), expnms = c('x1', 'x2'),
                       data=dat, q=4, B=10, seed=125)
dat2 <- data.frame(psi=seq(1,4, by=0.1))
summary(msm.predict(obj))
summary(msm.predict(obj, newdata=dat2))
```

msm_fit *Fitting marginal structural model (MSM) within quantile g-computation*

Description

This is an internal function called by qgcomp, qgcomp.glm.boot, and qgcomp.glm.noboot, but is documented here for clarity. Generally, users will not need to call this function directly.

Usage

```
msm_fit(
  f,
  qdata,
  intvals,
  expnms,
  rr = TRUE,
  main = TRUE,
  degree = 1,
  id = NULL,
  weights,
  bayes = FALSE,
  MCsize = nrow(qdata),
  hasintercept = TRUE,
  ...
)
```

Arguments

f	an r formula representing the conditional model for the outcome, given all exposures and covariates. Interaction terms that include exposure variables should be represented via the AsIs function
qdata	a data frame with quantized exposures
intvals	sequence, the sequence of integer values that the joint exposure is 'set' to for estimating the msm. For quantile g-computation, this is just 0:(q-1), where q is the number of quantiles of exposure.
expnms	a character vector with the names of the columns in qdata that represent the exposures of interest (main terms only!)
rr	logical, estimate log(risk ratio) (family='binomial' only)
main	logical, internal use: produce estimates of exposure effect (psi) and expected outcomes under g-computation and the MSM
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic. Default=1)
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	"case weights" - passed to the "weight" argument of glm or bayesglm
bayes	use underlying Bayesian model (arm package defaults). Results in penalized parameter estimation that can help with very highly correlated exposures. Note: this does not lead to fully Bayesian inference in general, so results should be interpreted as frequentist.
MCsize	integer: sample size for simulation to approximate marginal zero inflated model parameters. This can be left small for testing, but should be as large as needed to reduce simulation error to an acceptable magnitude (can compare psi coefficients for linear fits with qgcomp.zi.noboot to gain some intuition for the level of expected simulation error at a given value of MCsize)

hasintercept	(logical) does the model have an intercept?
...	arguments to glm (e.g. family)

Details

This function first computes expected outcomes under hypothetical interventions to simultaneously set all exposures to a specific quantile. These predictions are based on g-computation, where the exposures are 'quantized', meaning that they take on ordered integer values according to their ranks, and the integer values are determined by the number of quantile cutpoints used. The function then takes these expected outcomes and fits an additional model (a marginal structural model) with the expected outcomes as the outcome and the intervention value of the exposures (the quantile integer) as the exposure. Under causal identification assumptions and correct model specification, the MSM yields a causal exposure-response representing the incremental change in the expected outcome given a joint intervention on all exposures.

See Also

qgcomp.glm.boot, and qgcomp

Examples

```
set.seed(50)
dat <- data.frame(y=runif(200), x1=runif(200), x2=runif(200), z=runif(200))
X <- c('x1', 'x2')
qdat <- quantize(dat, X, q=4)$data
mod <- msm_fit(f=y ~ z + x1 + x2 + I(x1*x2),
        expnms = c('x1', 'x2'), qdata=qdat, intvals=1:4, bayes=FALSE)
summary(mod$fit) # outcome regression model
summary(mod$msmfit) # msm fit (variance not valid - must be obtained via bootstrap)
```

---

msm_multinomial_fit *Fitting marginal structural model (MSM) within quantile g-computation*

Description

This is an internal function called by qgcomp.multinomial.boot, but is documented here for clarity. Generally, users will not need to call this function directly.

Usage

```
msm_multinomial_fit(
  f,
  qdata,
  intvals,
  expnms,
  main = TRUE,
  degree = 1,
```

```
  id = NULL,
  weights,
  bayes = FALSE,
  MCsize = nrow(qdata),
  hasintercept = TRUE,
  ...
)
```

Arguments

f	an r formula representing the conditional model for the outcome, given all exposures and covariates. Interaction terms that include exposure variables should be represented via the AsIs function
qdata	a data frame with quantized exposures
intvals	sequence, the sequence of integer values that the joint exposure is 'set' to for estimating the msm. For quantile g-computation, this is just 0:(q-1), where q is the number of quantiles of exposure.
expnms	a character vector with the names of the columns in qdata that represent the exposures of interest (main terms only!)
main	logical, internal use: produce estimates of exposure effect (psi) and expected outcomes under g-computation and the MSM
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic. Default=1)
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	"case weights" - passed to the "weight" argument of multinom
bayes	use underlying Bayesian model (arm package defaults). Results in penalized parameter estimation that can help with very highly correlated exposures. Note: this does not lead to fully Bayesian inference in general, so results should be interpreted as frequentist.
MCsize	integer: sample size for simulation to approximate marginal zero inflated model parameters. This can be left small for testing, but should be as large as needed to reduce simulation error to an acceptable magnitude (can compare psi coefficients for linear fits with qgcomp.zi.noboot to gain some intuition for the level of expected simulation error at a given value of MCsize)
hasintercept	(logical) does the model have an intercept?
...	arguments to nnet::multinom

Details

This function first computes expected outcomes under hypothetical interventions to simultaneously set all exposures to a specific quantile. These predictions are based on g-computation, where the exposures are 'quantized', meaning that they take on ordered integer values according to their ranks, and the integer values are determined by the number of quantile cutpoints used. The function then takes these expected outcomes and fits an additional model (a marginal structural model) with the

expected outcomes as the outcome and the intervention value of the exposures (the quantile integer) as the exposure. Under causal identification assumptions and correct model specification, the MSM yields a causal exposure-response representing the incremental change in the expected outcome given a joint intervention on all exposures.

See Also

qgcomp.glm.boot, and qgcomp

Examples

```
data("metals") # from qgcomp package
# create categorical outcome from the existing continuous outcome (usually, one will already exist)
metals$ycat = factor(quantize(metals, "y",q=4)$data$y, levels=c("0", "1", "2", "3"),
                     labels=c("cct", "ccg", "aat", "aag"))
# restrict to smaller dataset for simplicity
smallmetals = metals[,c("ycat", "arsenic", "lead", "cadmium", "mage35")]

### 1: Define mixture and underlying model ####
mixture = c("arsenic", "lead", "cadmium")
f0 = ycat ~ arsenic + lead + cadmium # the multinomial model
# (be sure that factor variables are properly coded ahead of time in the dataset)
qdat <- quantize(smallmetals, mixture, q=4)$data
mod <- msm_multinomial_fit(f0,
        expnms = mixture, qdata=qdat, intvals=1:4, bayes=FALSE)
summary(mod$fit) # outcome regression model
summary(mod$msmfit) # msm fit (variance not valid - must be obtained via bootstrap)
```

---

plot.qgcompfit *Default plotting method for a qgcompfit object*

Description

Plot a quantile g-computation object. For qgcomp.glm.noboot, this function will create a butterfly plot of weights. For qgcomp.glm.boot, this function will create a box plot with smoothed line overlaying that represents a non-parametric fit of a model to the expected outcomes in the population at each quantile of the joint exposures (e.g. '1' represents 'at the first quantile for every exposure')

Usage

```
## S3 method for class 'qgcompfit'
plot(
  x,
  suppressprint = FALSE,
  geom_only = FALSE,
  pointwisebars = TRUE,
  modelfitline = TRUE,
  modelband = TRUE,
  flexfit = TRUE,
```

```
  pointwiseref = ceiling(x$q/2),
  ...
)

## S3 method for class 'qgcompmultfit'
plot(
  x,
  suppressprint = FALSE,
  pointwisebars = TRUE,
  modelfitline = TRUE,
  modelband = TRUE,
  flexfit = TRUE,
  pointwiseref = ceiling(x$q/2),
  ...
)
```

Arguments

x	"qgcompfit" object from qgcomp.glm.noboot, qgcomp.glm.boot, qgcomp.cox.noboot, qgcomp.cox.boot, qgcomp.zi.noboot or qgcomp.zi.boot functions
suppressprint	If TRUE, suppresses the plot, rather than printing it by default (it can be saved as a ggplot2 object (or list of ggplot2 objects if x is from a zero- inflated model) and used programmatically) (default = FALSE)
geom_only	If TRUE, returns only the geometry (i.e. does not contain the entire plot object). Used for overlays. Only used for .ee and .boot methods. (default = FALSE)
pointwisebars	(boot.gcomp only) If TRUE, adds 95% error bars for pointwise comparisons of E(Y|joint exposure) to the smooth regression line plot
modelfitline	(boot.gcomp only) If TRUE, adds fitted (MSM) regression line of E(Y|joint exposure) to the smooth regression line plot
modelband	If TRUE, adds 95% prediction bands for E(Y|joint exposure) (the MSM fit)
flexfit	(boot.gcomp only) if TRUE, adds flexible interpolation of predictions from underlying (conditional) model
pointwiseref	(boot.gcomp only) integer: which category of exposure (from 1 to q) should serve as the referent category for pointwise comparisons? (default=1)
...	unused

Functions

- plot(qgcompmultfit): Plot method for qgcomp multinomial fits

See Also

qgcomp.glm.noboot, qgcomp.glm.boot, and qgcomp

Examples

```
set.seed(12)
dat <- data.frame(x1=(x1 <- runif(100)), x2=runif(100), x3=runif(100), z=runif(100),
                  y=runif(100)+x1+x1^2)
ft <- qgcomp.glm.noboot(y ~ z + x1 + x2 + x3, expnms=c('x1','x2','x3'), data=dat, q=4)
ft
# display weights
plot(ft)
# examining fit
plot(ft$fit, which=1) # residual vs. fitted is not straight line!
## Not run:

# using non-linear outcome model
ft2 <- qgcomp.glm.boot(y ~ z + x1 + x2 + x3 + I(x1*x1), expnms=c('x1','x2','x3'),
data=dat, q=4, B=10)
ft2
plot(ft2$fit, which=1) # much better looking fit diagnostics suggests
# it is better to include interaction term for x
plot(ft2) # the msm predictions don't match up with a smooth estimate
# of the expected outcome, so we should consider a non-linear MSM

# using non-linear marginal structural model
ft3 <- qgcomp.glm.boot(y ~ z + x1 + x2 + x3 + I(x1*x1), expnms=c('x1','x2','x3'),
data=dat, q=4, B=10, degree=2)
# plot(ft3$fit, which=1) - not run - this is identical to ft2 fit
plot(ft3) # the MSM estimates look much closer to the smoothed estimates
# suggesting the non-linear MSM fits the data better and should be used
# for inference about the effect of the exposure

# binary outcomes, logistic model with or without a log-binomial marginal
structural model
dat <- data.frame(y=rbinom(100,1,0.5), x1=runif(100), x2=runif(100), z=runif(100))
fit1 <- qgcomp.glm.boot(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
         data=dat, q=9, B=100, rr=FALSE)
fit2 <- qgcomp.glm.boot(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
         data=dat, q=9, B=100, rr=TRUE)
plot(fit1)
plot(fit2)
# Using survival data ()
set.seed(50)
N=200
dat <- data.frame(time=(tmg <- pmin(.1,rweibull(N, 10, 0.1))),
                d=1.0*(tmg<0.1), x1=runif(N), x2=runif(N), z=runif(N))
expnms=paste0("x", 1:2)
f = survival::Surv(time, d)~x1 + x2
(fit1 <- survival::coxph(f, data = dat))
# non-bootstrap method to get a plot of weights
(obj <- qgcomp.cox.noboot(f, expnms = expnms, data = dat))
plot(obj)

# bootstrap method to get a survival curve
# this plots the expected survival curve for the underlying (conditional) model
```

```
# as well as the expected survival curve for the MSM under the following scenarios:
#  1) highest joint exposure category
#  2) lowest joint exposure category
#  3) average across all exposure categories
# differences between the MSM and conditional fit suggest that the MSM is not flexible
# enough to accomodate non-linearities in the underlying fit (or they may simply signal that
# MCSize should be higher). Note that if linearity
# is assumed in the conditional model, the MSM will typically also appear linear and
# will certainly appear linear if no non-exposure covariates are included in the model
# not run (slow when using boot version to proper precision)
(obj2 <- qgcomp.cox.boot(f, expnms = expnms, data = dat, B=10, MCsize=2000))
plot(obj2)

## End(Not run)
```

`pointwisebound.boot` *Estimating pointwise comparisons for qgcomp.glm.boot objects*

Description

Calculates: expected outcome (on the link scale), mean difference (link scale) and the standard error of the mean difference (link scale) for pointwise comparisons

Usage

```
pointwisebound.boot(x, pointwiseref = 1, alpha = 0.05)
```

Arguments

`x`	"qgcompfit" object from `qgcomp.glm.boot`,
`pointwiseref`	referent quantile (e.g. 1 uses the lowest joint-exposure category as the referent category for calculating all mean differences/standard deviations)
`alpha`	alpha level for confidence intervals

Details

The comparison of interest following a qgcomp fit is often comparisons of model predictions at various values of the joint-exposures (e.g. expected outcome at all exposures at the 1st quartile vs. the 3rd quartile). The expected outcome at a given joint exposure, and marginalized over non-exposure covariates (W), is given as E(Y^s|S) = sum_w E_w(Y|S,W)Pr(W) = sum_i E(Y_i|S) where Pr(W) is the emprical distribution of W and S takes on integer values 0 to q-1. Thus, comparisons are of the type E_w(Y|S=s) - E_w(Y|S=s2) where s and s2 are two different values of the joint exposures (e.g. 0 and 2). This function yields E_w(Y|S) as well as E_w(Y|S=s) - E_w(Y|S=p) where s is any value of S and p is the value chosen via "pointwise ref" - e.g. for binomial variables this will equal the risk/ prevalence difference at all values of S, with the referent category S=p-1. The standard error of E(Y|S=s) - E(Y|S=p) is calculated from the bootstrap covariance matrix of E_w(Y|S), such that the standard error for E_w(Y|S=s) - E_w(Y|S=p) is given by

Var(E_w(Y|S=s)) + - Var(E_w(Y|S=p)) - 2*Cov(E_w(Y|S=p), - E_w(Y|S=s))

This is used to create pointwise confidence intervals. Note that this differs slightly from the pointwisebound.noboot function, which estimates the variance of the conditional regression line given by E(Y|S,W=w), where w is a vector of medians of W (i.e. predictions are made at the median value of all covariates).

Value

A data frame containing

linpred: The linear predictor from the marginal structural model

rr/or/mean.diff: The canonical effect measure (risk ratio/odds ratio/mean difference) for the marginal structural model link

se....: the stndard error of the effect measure

ul..../ll....: Confidence bounds for the effect measure, and bounds centered at the linear predictor (for plotting purposes)

See Also

qgcomp.glm.boot, pointwisebound.noboot

Examples

```
set.seed(12)
## Not run:
n=100
# non-linear model for continuous outcome
dat <- data.frame(x1=(x1 <- runif(100)), x2=runif(100), x3=runif(100), z=runif(100),
                  y=runif(100)+x1+x1^2)
ft <- qgcomp.glm.boot(y ~ z + x1 + x2 + x3, expnms=c('x1','x2','x3'), data=dat, q=10)
pointwisebound.boot(ft, alpha=0.05, pointwiseref=3)

## End(Not run)
```

---

pointwisebound.noboot *Estimating pointwise comparisons for qgcomp.glm.noboot objects*

Description

Calculates: expected outcome (on the link scale), mean difference (link scale) and the standard error of the mean difference (link scale) for pointwise comparisons

Usage

```
pointwisebound.noboot(x, alpha = 0.05, pointwiseref = 1)
```

Arguments

x	"qgcompfit" object from qgcomp.glm.noboot,
alpha	alpha level for confidence intervals
pointwiseref	referent quantile (e.g. 1 uses the lowest joint-exposure category as the referent category for calculating all mean differences/standard deviations)

Details

The comparison of interest following a qgcomp fit is often comparisons of model predictions at various values of the joint-exposures (e.g. expected outcome at all exposures at the 1st quartile vs. the 3rd quartile). The expected outcome at a given joint exposure and at a given level of non-exposure covariates (W=w) is given as E(Y|S,W=w), where S takes on integer values 0 to q-1. Thus, comparisons are of the type E(Y|S=s,W=w) - E(Y|S=s2,W=w) where s and s2 are two different values of the joint exposures (e.g. 0 and 2). This function yields E(Y|S,W=w) as well as E(Y|S=s,W=w) - E(Y|S=p,W=w) where s is any value of S and p is the value chosen via "pointwise ref" - e.g. for binomial variables this will equal the risk/ prevalence difference at all values of S, with the referent category S=p-1. For the non-boostrapped version of quantile g-computation (under a linear model). Note that w is taken to be the referent level of covariates so that if meaningful values of E(Y|S,W=w) and E(Y|S=s,W=w) - E(Y|S=p,W=w) are desired, then it is advisable to set the referent levels of W to meaningful values. This can be done by, e.g. centering continuous age so that the predictions are made at the population mean age, rather than age 0.

Note that function only works with standard "qgcompfit" objects from qgcomp.glm.noboot or qgcomp.glm.ee (so it doesn't work with zero inflated, hurdle, or Cox models)

Variance for the overall effect estimate is given by: $transpose(G)Cov(\beta)G$

Where the "gradient vector" G is given by

$$G = [\partial(f(\beta))/\partial\beta_1 = 1, ..., \partial(f(\beta))/\partial\beta_3 k = 1]$$

$f(\beta) = \sum_i^p \beta_i$, and $\partial y/\partial x$ denotes the partial derivative/gradient. The vector G takes on values that equal the difference in quantiles of S for each pointwise comparison (e.g. for a comparison of the 3rd vs the 5th category, G is a vector of 2s)

This variance is used to create pointwise confidence intervals via a normal approximation: (e.g. upper 95% CI = psi + variance*1.96)

Value

A data frame containing

hx: The "partial" linear predictor $\beta_0 + \psi \sum_j X_j^q w_j$, or the effect of the mixture + intercept after conditioning out any confounders. This is similar to the h(x) function in bkmr. This is not a full prediction of the outcome, but only the partial outcome due to the intercept and the confounders

rr/or/mean.diff: The canonical effect measure (risk ratio/odds ratio/mean difference) for the marginal structural model link

se....: the stndard error of the effect measure

ul..../ll....: Confidence bounds for the effect measure

See Also

qgcomp.glm.noboot, pointwisebound.boot

Examples

```
set.seed(12)
## Not run:
n = 100
dat <- data.frame(x1=(x1 <- runif(n)), x2=(x2 <- runif(n)),
                  x3=(x3 <- runif(n)), z=(z <- runif(n)),
                  y=rnorm(n)+x1 + x2 - x3 +z)
# linear model for continuous outcome
ft <- qgcomp.glm.noboot(y ~ z + x1 + x2 + x3,
       expnms=c('x1','x2','x3'), data=dat, q=10)
ft2 <- qgcomp.glm.boot(y ~ z + x1 + x2 + x3,
        expnms=c('x1','x2','x3'), data=dat, q=10)
pointwisebound.noboot(ft, alpha=0.05, pointwiseref=3)
pointwisebound.boot(ft2, alpha=0.05, pointwiseref=3)
dat <- data.frame(x1=(x1 <- runif(n)), x2=(x2 <- runif(n)),
                   x3=(x3 <- runif(n)), z=(z <- runif(n)),
                  y=rbinom(n, 1, 1/(1+exp(-(x1 + x2 - x3 +z)))))
# glms for binary outcome, centering covariate to a potentially more meaningful value
dat$zcen = dat$z - mean(dat$z)
ft <- qgcomp.glm.noboot(y ~ zcen + x1 + x2 + x3,
        expnms=c('x1','x2','x3'), data=dat, q=10, family=binomial())
ft2 <- qgcomp.glm.boot(y ~ zcen + x1 + x2 + x3,
        expnms=c('x1','x2','x3'), data=dat, q=10, family=binomial())
pointwisebound.noboot(ft, alpha=0.05, pointwiseref=3)
pointwisebound.boot(ft2, alpha=0.05, pointwiseref=3)
dat$z = as.factor(sample(1:3, n, replace=TRUE))
ftf <- qgcomp.glm.noboot(y ~ zcen + x1 + x2 + x3,
       expnms=c('x1','x2','x3'), data=dat, q=10, family=binomial())
pointwisebound.noboot(ftf, alpha=0.05, pointwiseref=3)

## End(Not run)
```

`predict.qgcompfit` *Default prediction method for a qgcompfit object (non-survival outcomes only)*

Description

get predicted values from a qgcompfit object, or make predictions in a new set of data based on the qgcompfit object. Note that when making predictions from an object from qgcomp.glm.boot, the predictions are made from the (conditional) g-computation model rather than the marginal structural model. Predictions from the marginal structural model can be obtained via msm.predict. Note that this function accepts non-quantized exposures in "newdata" and automatically quantizes them according to the quantile cutpoints in the original fit.

Usage

```
## S3 method for class 'qgcompfit'
predict(object, expnms = NULL, newdata = NULL, type = "response", ...)
```

Arguments

object	"qgcompfit" object from qgcomp.glm.noboot, qgcomp.glm.boot, qgcomp.zi.noboot, or qgcomp.zi.bootfunctions
expnms	character vector of exposures of interest
newdata	(optional) new set of data with all predictors from "qgcompfit" object
type	(from predict.glm) the type of prediction required. The default is on the scale of the linear predictors; the alternative "response" is on the scale of the response variable. Thus for a default binomial model the default predictions are of log-odds (probabilities on logit scale) and type = "response" gives the predicted probabilities. The "terms" option returns a matrix giving the fitted values of each term in the model formula on the linear predictor scale.
...	arguments to predict.glm

Examples

```
set.seed(50)
dat <- data.frame(y=runif(50), x1=runif(50), x2=runif(50), z=runif(50))
obj1 <- qgcomp.glm.noboot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2)
obj2 <- qgcomp.glm.boot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, B=10, seed=125)
set.seed(52)
dat2 <- data.frame(y=runif(50), x1=runif(50), x2=runif(50), z=runif(50))
summary(predict(obj1, expnms = c('x1', 'x2'), newdata=dat2))
summary(predict(obj2, expnms = c('x1', 'x2'), newdata=dat2))
```

---

print.qgcompfit *Default printing method for a qgcompfit object*

Description

Gives variable output depending on whether qgcomp.glm.noboot or qgcomp.glm.boot is called. For qgcomp.glm.noboot will output final estimate of joint exposure effect (similar to the 'index' effect in weighted quantile sums), as well as estimates of the 'weights' (standardized coefficients). For qgcomp.glm.boot, the marginal effect is given, but no weights are reported since this approach generally incorporates non-linear models with interaction terms among exposures, which preclude weights with any useful interpretation.

Usage

```
## S3 method for class 'qgcompfit'
print(x, showweights = TRUE, ...)
```

Arguments

x	"qgcompfit" object from qgcomp, qgcomp.glm.noboot or qgcomp.glm.boot function
showweights	logical: should weights be printed, if estimated?
...	unused

See Also

qgcomp.glm.noboot, qgcomp.glm.boot, and qgcomp

Examples

```
set.seed(50)
dat <- data.frame(y=runif(50), x1=runif(50), x2=runif(50), z=runif(50))
obj1 <- qgcomp.glm.noboot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2)
obj2 <- qgcomp.glm.boot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, B=10, seed=125)
# does not need to be explicitly called, but included here for clarity
print(obj1)
print(obj2)
```

---

qgcomp *Quantile g-computation for continuous, binary, count, and censored survival outcomes*

Description

This function automatically selects between qgcomp.glm.noboot, qgcomp.glm.boot, qgcomp.cox.noboot, and qgcomp.cox.boot for the most efficient approach to estimate the average expected change in the (log) outcome per quantile increase in the joint exposure to all exposures in expnms', given the underlying model. For e comp.glm.noboot" will be called to fit the model. Non-linear terms or requesting the risk ratio for binomial outcomes will result in the qgcomp.glm.boot function being called. For a given linear model, boot and noboot versions will give identical inference, though when using survival outcomes, the 'boot' version uses simulation based inference, which can vary from the 'noboot' version due to simulation error (which can be minimized via setting the MCsize parameter very large - see qgcomp.cox.boot for details).

Usage

```
qgcomp(f, data = data, family = gaussian(), rr = TRUE, ...)
```

Arguments

f	R style formula (may include survival outcome via Surv)
data	data frame
family	gaussian(), binomial(), cox(), poisson() (works as argument to 'family' parameter in glm' or 'dist' parameter in zeroinfl)

rr	logical: if using binary outcome and rr=TRUE, qgcomp.glm.boot will estimate risk ratio rather than odds ratio. Note, to get population average effect estimates for a binary outcome, set rr=TRUE (default: ORs are generally not of interest as population average effects, so if rr=FALSE then a conditional OR will be estimated, which cannot be interpreted as a population average effect
...	arguments to qgcomp.glm.noboot or qgcomp.glm.boot (e.g. q) or glm

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and possibly information on the marginal structural model (msmfit) used to estimate the final effect estimates (qgcomp.glm.boot, qgcomp.cox.boot only). If appropriate, weights are also reported, which represent the proportion of a directional (positive/negative) effect that is accounted for by each exposure.

See Also

qgcomp.glm.noboot, qgcomp.glm.boot, qgcomp.cox.noboot, qgcomp.cox.boot qgcomp.zi.noboot and qgcomp.zi.boot (qgcomp is just a wrapper for these functions)

Examples

```
set.seed(50)
dat <- data.frame(y=runif(50), x1=runif(50), x2=runif(50), z=runif(50))
qgcomp.glm.noboot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2)
qgcomp.glm.boot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, B=10, seed=125)
# automatically selects appropriate method
qgcomp(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2)
# note for binary outcome this will choose the risk ratio (and bootstrap methods) by default
dat <- data.frame(y=rbinom(100, 1, 0.5), x1=runif(100), x2=runif(100), z=runif(100))
## Not run:
qgcomp.glm.noboot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=binomial())
set.seed(1231)
qgcomp.glm.boot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=binomial())
set.seed(1231)
qgcomp(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=binomial())

# automatically selects appropriate method when specifying rr or degree explicitly
qgcomp(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=binomial(), rr=FALSE)
qgcomp(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=binomial(), rr=TRUE)
qgcomp(y ~ z + factor(x1) + factor(x2), degree=2, expnms = c('x1', 'x2'), data=dat, q=4,
family=binomial())

#survival objects
set.seed(50)
N=200
dat <- data.frame(time=(tmg <- pmin(.1,rweibull(N, 10, 0.1))),
                d=1.0*(tmg<0.1), x1=runif(N), x2=runif(N), z=runif(N))
expnms=paste0("x", 1:2)
f = survival::Surv(time, d)~x1 + x2
qgcomp(f, expnms = expnms, data = dat)
```

```
# note if B or MCsize are set but the model is linear, an error will result
try(qgcomp(f, expnms = expnms, data = dat, B1=, MCsize))
# note that in the survival models, MCsize should be set to a large number
#  such that results are repeatable (within an error tolerance such as 2 significant digits)
# if you run them under different  seed values
f = survival::Surv(time, d)~x1 + x2 + x1:x2
qgcomp(f, expnms = expnms, data = dat, B=10, MCsize=100)

## End(Not run)
```

`qgcomp.cch.noboot` *Quantile g-computation for survival outcomes in a case-cohort design under linearity/additivity*

Description

This function performs quantile g-computation in a survival setting with case-cohort sampling. The approach estimates the covariate-conditional hazard ratio for a joint change of 1 quantile in each exposure variable specified in expnms parameter

Usage

```
qgcomp.cch.noboot(
  f,
  data,
  subcoh = NULL,
  id = NULL,
  cohort.size = NULL,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  weights,
  cluster = NULL,
  alpha = 0.05,
  ...
)
```

Arguments

`f`	R style survival formula, which includes `Surv` in the outcome definition. E.g. `Surv(time,event) ~ exposure`. Offset terms can be included via `Surv(time,event) ~ exposure + offset(z)`
`data`	data frame
`subcoh`	(From `cch` help) Vector of indicators for subjects sampled as part of the sub-cohort. Code 1 or TRUE for members of the sub-cohort, 0 or FALSE for others. If data is a data frame then subcoh may be a one-sided formula.

id	(From cch help) Vector of unique identifiers, or formula specifying such a vector.
cohort.size	(From cch help) Vector with size of each stratum original cohort from which subcohort was sampled
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints. See examples for how you might use this in case-cohort studies.
weights	Not used here (argument will be ignored)
cluster	Not used here (argument will be ignored)
alpha	alpha level for confidence limit calculation
...	arguments to cch (e.g. robust, method, stratum - see examples)

Details

For survival outcomes (as specified using methods from the survival package), this yields a conditional log hazard ratio representing a change in the expected conditional hazard (conditional on covariates) from increasing every exposure by 1 quantile. In general, this quantity quantity is not equivalent to marginal g-computation estimates. Hypothesis test statistics and 95% confidence intervals are based on using the delta method estimate variance of a linear combination of random variables.

Note that this closely follows the cch function in the survival package by Terry Therneau, and is restricted to the methods used in that function, which may not address all extant methods for case-cohort studies.

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the weights/standardized coefficients in the positive (pos.weights) and negative (neg.weights) directions.

See Also

Other qgcomp_methods: qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boot(), qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
set.seed(50)
N=500
# cohort analysis
dat <- data.frame(id = 1:N, time=(tmg <- pmin(.1,rweibull(N, 10, 0.1))),
                d=1.0*(tmg<0.1), x1=runif(N), x2=runif(N), z=rbinom(N, 1, 0.5))
expnms=paste0("x", 1:2)
f1 = survival::Surv(time, d)~x1 + x2 + z
(fit1 <- survival::coxph(f1, data = dat))
(obj <- qgcomp.cox.noboot(f1, expnms = expnms, data = dat))
f1s = survival::Surv(time, d)~x1 + x2 + survival::strata(z)
(fit1s <- survival::coxph(f1s, data = dat))
(objs <- qgcomp.cox.noboot(f1s, expnms = expnms, data = dat))
#### now doing a case-cohort analysis
# 1) sampling simple case-cohort data
dat$subcohort = 1:nrow(dat) %in% sort(sample(1:nrow(dat), 100))
caco_dat = dat[dat$subcohort | dat$d,]
dim(caco_dat)
dim(dat)

# getting quantile categories from the subcohort
qdata = quantize(caco_dat[caco_dat$subcohort,], expnms=expnms)
qdata$breaks
# 2) doing simple (unstratified) analysis
f2 = survival::Surv(time, d)~x1 + x2 + z
(obj2 <- qgcomp.cch.noboot(f2, expnms = expnms, breaks = qdata$breaks,
         data = caco_dat, subcoh = ~ subcohort, id = ~id, cohort.size=N))
obj2$fit

### doing stratified analysis (if subcohort and/or cases are a stratified sample)
# 1) sampling stratified case-cohort data
sampfracs = c(.25, .75) # z=0 vs. z=1
nco = 100 # total subcohort members
nca = nrow(dat[dat$d==1,])
selected_ids = sort(c(sample(dat[dat$z==0, "id"], round(nco*sampfracs[1])),
                 sample(dat[dat$z==1, "id"],  round(nco*sampfracs[2]))))
selected_cases = sort(c(sample(dat[dat$d==1 & dat$z==0, "id"], round(nca*sampfracs[1])),
                 sample(dat[dat$d==1 & dat$z==1, "id"], round(nca*sampfracs[1]))))
dat$subcohort = dat$id %in% selected_ids
dat$selectedcases = dat$id %in% selected_cases
caco_dat_strat = dat[dat$subcohort | dat$selectedcases,]
dim(caco_dat_strat)
dim(dat)

# getting quantile categories from the subcohort by differential sampling across strata
subco_strat = caco_dat_strat[caco_dat_strat$subcohort,]
z_stratum_sizes = table(dat$z)
z_stratum_sizes_caco = table(subco_strat$z)
sampweights = z_stratum_sizes/z_stratum_sizes_caco
sampweightsn = sampweights/(min(sampweights))

# now oversample the undersampled into a dataset used to create cutpoints
```

```
wtdcutids = data.frame(id=sort(c(sample(subco_strat[subco_strat$z==0,"id"],
                sampweightsn[1]*z_stratum_sizes_caco[1], replace=TRUE),
                sample(subco_strat[subco_strat$z==1,"id"],
                sampweightsn[2]*z_stratum_sizes_caco[2]))))
cutdata = merge(wtdcutids,subco_strat, all.x=TRUE)
qdata_strat = quantize(cutdata, expnms=expnms)
qdata_strat$breaks
f2s = survival::Surv(time, d)~x1 + x2
(obj2s <- qgcomp.cch.noboot(f2s, expnms = expnms, breaks = qdata_strat$breaks,
         data = caco_dat_strat, subcoh = ~ subcohort, id = ~id,
         stratum=~z,
         cohort.size=z_stratum_sizes, method="I.Borgan"))
obj2s$fit
```

---

qgcomp.cox.boot *Quantile g-computation for survival outcomes*

Description

This function yields population average effect estimates for (possibly right censored) time-to event outcomes

Usage

```
qgcomp.cox.boot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  cluster = NULL,
  alpha = 0.05,
  B = 200,
  MCsize = 10000,
  degree = 1,
  seed = NULL,
  parallel = FALSE,
  parplan = FALSE,
  ...
)
```

Arguments

f	R style survival formula, which includes Surv in the outcome definition. E.g. Surv(time,event) ~ exposure. Offset terms can be included via Surv(time,event) ~ exposure + offset(z)
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster). Note that qgcomp.cox.noboot will not produce cluster-appropriate standard errors. qgcomp.cox.boot can be used for this, which will use bootstrap sampling of clusters/individuals to estimate cluster-appropriate standard errors via bootstrapping.
weights	"case weights" - passed to the "weight" argument of coxph
cluster	not implemented (here, "id" plays the same role as cluster plays in cqgcomp.cox.noboot)
alpha	alpha level for confidence limit calculation
B	integer: number of bootstrap iterations (this should typically be >=200, though it is set lower in examples to improve run-time).
MCsize	integer: sample size for simulation to approximate marginal hazards ratios (if < sample size, then set to sample size). Note that large values will slow down the fitting, but will result in higher accuracy - if you run the function multiple times you will see that results vary due to simulation error. Ideally, MCsize would be set such that simulation error is negligible in the precision reported (e.g. if you report results to 2 decimal places, then MCsize should be set high enough that you consistenty get answers that are the same to 2 decimal places).
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic.
seed	integer or NULL: random number seed for replicable bootstrap results
parallel	logical (default FALSE): use future package to speed up bootstrapping
parplan	(logical, default=FALSE) automatically set future::plan to plan(multisession) (and set to existing plan, if any, after bootstrapping)
...	arguments to coxph

Details

qgcomp.cox.boot' estimates the log(hazard ratio) per quantile increase in the joint exposure to all expo
pnms'. This function uses g-computation to estimate the parameters of a marginal structural model for the population average effect of increasing all exposures in 'expnms' by a single quantile. This

approach involves specifying an underlying conditional outcome model, given all exposures of interest (possibly with non-linear basis function representations such as splines or product terms) and confounders or covariates of interest. This model is fit first, which is used to generate expected outcomes at each quantile of all exposures, which is then used in a second model to estimate a population average dose-response curve that is linear or follows a simple polynomial function. See section on MCSize below

Test statistics and confidence intervals are based on a non-parametric bootstrap, using the standard deviation of the bootstrap estimates to estimate the standard error. The bootstrap standard error is then used to estimate Wald-type confidence intervals. Note that no bootstrapping is done on estimated quantiles of exposure, so these are treated as fixed quantities

MCSize is crucial to get accurate point estimates. In order to get marginal estimates of the population hazard under different values of the joint exposure at a given quantile for all exposures in `expnms`, `qgcomp.cox.boot` uses Monte Carlo simulation to generate outcomes implied by the underlying conditional model and then fit a separate (marginal structural) model to those outcomes. In order to get accurate results that don't vary much from run-to-run of this approach, MCsize must be set large enough so that results are stable across runs according to a pre-determined precision (e.g. 2 significant digits).

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the marginal structural model (msmfit) used to estimate the final effect estimates.

See Also

Other qgcomp_methods: `qgcomp.cch.noboot()`, `qgcomp.cox.noboot()`, `qgcomp.glm.boot()`, `qgcomp.glm.ee()`, `qgcomp.glm.noboot()`, `qgcomp.hurdle.boot()`, `qgcomp.hurdle.noboot()`, `qgcomp.multinomial.boot()`, `qgcomp.multinomial.noboot()`, `qgcomp.partials()`, `qgcomp.zi.boot()`, `qgcomp.zi.noboot()`

Examples

```
set.seed(50)
N=200
dat <- data.frame(time=(tmg <- pmin(.1,rweibull(N, 10, 0.1))),
                d=1.0*(tmg<0.1), x1=runif(N), x2=runif(N), z=runif(N))
expnms=paste0("x", 1:2)
f = survival::Surv(time, d)~x1 + x2
(fit1 <- survival::coxph(f, data = dat))
(obj <- qgcomp.cox.noboot(f, expnms = expnms, data = dat))
## Not run:
# not run (slow when using boot version to proper precision)
(obj2 <- qgcomp.cox.boot(f, expnms = expnms, data = dat, B=10, MCsize=20000))

# weighted analysis

# using future package, marginalizing over confounder z
(obj3 <- qgcomp.cox.boot(survival::Surv(time, d)~x1 + x2 + z, expnms = expnms, data = dat,
                         B=1000, MCsize=20000, parallel=TRUE, parplan=TRUE))
```

```
# non-constant hazard ratio, non-linear terms
(obj4 <- qgcomp.cox.boot(survival::Surv(time, d)~factor(x1) + splines::bs(x2) + z,
                         expnms = expnms, data = dat,
                         B=1000, MCsize=20000, parallel=FALSE, degree=1))

# weighted analysis
dat$w = runif(N)
(objw1 <- qgcomp.cox.noboot(f, expnms = expnms, data = dat, weights=w))
(objw2 <- qgcomp.cox.boot(f, expnms = expnms, data = dat, weights=w, B=5, MCsize=20000))

## End(Not run)
```

`qgcomp.cox.noboot` *Quantile g-computation for survival outcomes under linearity/additivity*

Description

This function performs quantile g-computation in a survival setting. The approach estimates the covariate-conditional hazard ratio for a joint change of 1 quantile in each exposure variable specified in expnms parameter

Usage

```
qgcomp.cox.noboot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  cluster = NULL,
  alpha = 0.05,
  ...
)
```

Arguments

`f`	R style survival formula, which includes `Surv` in the outcome definition. E.g. `Surv(time,event) ~ exposure`. Offset terms can be included via `Surv(time,event) ~ exposure + offset(z)`
`data`	data frame
`expnms`	character vector of exposures of interest
`q`	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)

breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	"case weights" - passed to the "weight" argument of coxph
cluster	(from coxph function help) optional variable which clusters the observations, for the purposes of a robust variance. If present, it implies robust. This variable will normally be found in data.
alpha	alpha level for confidence limit calculation
...	arguments to coxph

Details

For survival outcomes (as specified using methods from the survival package), this yields a conditional log hazard ratio representing a change in the expected conditional hazard (conditional on covariates) from increasing every exposure by 1 quantile. In general, this quantity quantity is not equivalent to g-computation estimates. Hypothesis test statistics and 95% confidence intervals are based on using the delta estimate variance of a linear combination of random variables.

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the weights/standardized coefficients in the positive (pos.weights) and negative (neg.weights) directions.

See Also

qgcomp.cox.boot, qgcomp.glm.boot, and qgcomp

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boot(), qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
set.seed(50)
N=200
dat <- data.frame(time=(tmg <- pmin(.1,rweibull(N, 10, 0.1))),
                d=1.0*(tmg<0.1), x1=runif(N), x2=runif(N), z=runif(N))
expnms=paste0("x", 1:2)
f = survival::Surv(time, d)~x1 + x2
(fit1 <- survival::coxph(f, data = dat))
(obj <- qgcomp.cox.noboot(f, expnms = expnms, data = dat))
## Not run:

# weighted analysis
dat$w = runif(N)
qdata = quantize(dat, expnms=expnms)
```

```
(obj2 <- qgcomp.cox.noboot(f, expnms = expnms, data = dat, weight=w))
obj2$fit
survival::coxph(f, data = qdata$data, weight=w)

# not run: bootstrapped version is much slower
(obj2 <- qgcomp.cox.boot(f, expnms = expnms, data = dat, B=200, MCsize=20000))
# checking whether missing data causes an issue
dat$z[1:10] <- NA
(objzmiss <- qgcomp.cox.noboot(f, expnms = expnms, data = dat))
(objzmiss_alt <- qgcomp.cox.noboot(f, expnms = expnms, data = dat[,c(expnms, "time", "d")]))
set.seed(110)
(objzmiss2 <- qgcomp.cox.boot(f, expnms = expnms, data = dat, B=3, MCsize=100))
dat$x1[1:10] <- NA
(objx1miss <- qgcomp.cox.noboot(f, expnms = expnms, data = dat))
set.seed(110)
(objx1miss2 <- qgcomp.cox.boot(f, expnms = expnms, data = dat, B=3, MCsize=100))

## End(Not run)
```

---

qgcomp.glm.boot *Quantile g-computation for continuous and binary outcomes*

Description

This function estimates a dose-response parameter representing a one quantile increase in a set of exposures of interest. This model estimates the parameters of a marginal structural model (MSM) based on g-computation with quantized exposures. Note: this function allows non-linear and non-additive effects of individual components of the exposure, as well as non-linear joint effects of the mixture via polynomial basis functions, which increase the computational computational burden due to the need for non-parametric bootstrapping. qgcomp.boot is an equivalent function (slated for deprecation)

Usage

```
qgcomp.glm.boot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  B = 200,
  rr = TRUE,
  degree = 1,
  seed = NULL,
  bayes = FALSE,
```

```
  MCsize = nrow(data),
  parallel = FALSE,
  parplan = FALSE,
  ...
)

qgcomp.boot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  B = 200,
  rr = TRUE,
  degree = 1,
  seed = NULL,
  bayes = FALSE,
  MCsize = nrow(data),
  parallel = FALSE,
  parplan = FALSE,
  ...
)
```

Arguments

f	R style formula
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster). Note that qgcomp.glm.noboot will not produce cluster-appropriate standard errors. qgcomp.glm.boot can be used for this, which will use bootstrap sampling of clusters/individuals to estimate cluster-appropriate standard errors via bootstrapping.
weights	"case weights" - passed to the "weight" argument of glm or bayesglm
alpha	alpha level for confidence limit calculation

B	integer: number of bootstrap iterations (this should typically be >=200, though it is set lower in examples to improve run-time).
rr	logical: if using binary outcome and rr=TRUE, qgcomp.glm.boot will estimate risk ratio rather than odds ratio
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic (default = 1).
seed	integer or NULL: random number seed for replicable bootstrap results
bayes	use underlying Bayesian model (arm package defaults). Results in penalized parameter estimation that can help with very highly correlated exposures. Note: this does not lead to fully Bayesian inference in general, so results should be interpreted as frequentist.
MCsize	integer: sample size for simulation to approximate marginal zero inflated model parameters. This can be left small for testing, but should be as large as needed to reduce simulation error to an acceptable magnitude (can compare psi coefficients for linear fits with qgcomp.glm.noboot to gain some intuition for the level of expected simulation error at a given value of MCsize). This likely won't matter much in linear models, but may be important with binary or count outcomes.
parallel	use (safe) parallel processing from the future and future.apply packages
parplan	(logical, default=FALSE) automatically set future::plan to plan(multisession) (and set to existing plan, if any, after bootstrapping)
...	arguments to glm (e.g. family)

Details

Estimates correspond to the average expected change in the (log) outcome per quantile increase in the joint exposure to all exposures in 'expnms'. Test statistics and confidence intervals are based on a non-parametric bootstrap, using the standard deviation of the bootstrap estimates to estimate the standard error. The bootstrap standard error is then used to estimate Wald-type confidence intervals. Note that no bootstrapping is done on estimated quantiles of exposure, so these are treated as fixed quantities

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the marginal structural model (msmfit) used to estimate the final effect estimates.

See Also

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boot(), qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
set.seed(30)
# continuous outcome
dat <- data.frame(y=rnorm(100), x1=runif(100), x2=runif(100), z=runif(100))
# Conditional linear slope
qgcomp.glm.noboot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4, family=gaussian())
# Marginal linear slope (population average slope, for a purely linear,
#  additive model this will equal the conditional)
 ## Not run:
qgcomp.glm.boot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4,
  family=gaussian(), B=200) # B should be at least 200 in actual examples
# no intercept model
qgcomp.glm.boot(f=y ~ -1+z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4,
  family=gaussian(), B=200) # B should be at least 200 in actual examples

 # Note that these give different answers! In the first, the estimate is conditional on Z,
 # but in the second, Z is marginalized over via standardization. The estimates
 # can be made approximately the same by centering Z (for linear models), but
 # the conditional estimate will typically have lower standard errors.
 dat$z = dat$z - mean(dat$z)

# Conditional linear slope
qgcomp.glm.noboot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4, family=gaussian())
# Marginal linear slope (population average slope, for a purely linear,
#  additive model this will equal the conditional)

qgcomp.glm.boot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4,
  family=gaussian(), B=200) # B should be at least 200 in actual examples

# Population average mixture slope which accounts for non-linearity and interactions
qgcomp.glm.boot(y ~ z + x1 + x2 + I(x1^2) + I(x2*x1), family="gaussian",
 expnms = c('x1', 'x2'), data=dat, q=4, B=200)

# generally non-linear/non-addiive underlying models lead to non-linear mixture slopes
qgcomp.glm.boot(y ~ z + x1 + x2 + I(x1^2) + I(x2*x1), family="gaussian",
 expnms = c('x1', 'x2'), data=dat, q=4, B=200, deg=2)

# binary outcome
dat <- data.frame(y=rbinom(50,1,0.5), x1=runif(50), x2=runif(50), z=runif(50))

# Conditional mixture OR
qgcomp.glm.noboot(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
  data=dat, q=2)

#Marginal mixture OR (population average OR - in general, this will not equal the
# conditional mixture OR due to non-collapsibility of the OR)
qgcomp.glm.boot(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
  data=dat, q=2, B=3, rr=FALSE)

# Population average mixture RR
qgcomp.glm.boot(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
  data=dat, q=2, rr=TRUE, B=3)
```

```
# Population average mixture RR, indicator variable representation of x2
# note that I(x==...) operates on the quantile-based category of x,
# rather than the raw value
res = qgcomp.glm.boot(y ~ z + x1 + I(x2==1) + I(x2==2) + I(x2==3),
  family="binomial", expnms = c('x1', 'x2'), data=dat, q=4, rr=TRUE, B=200)
res$fit
plot(res)

# now add in a non-linear MSM
res2 = qgcomp.glm.boot(y ~ z + x1 + I(x2==1) + I(x2==2) + I(x2==3),
  family="binomial", expnms = c('x1', 'x2'), data=dat, q=4, rr=TRUE, B=200,
  degree=2)
res2$fit
res2$msmfit  # correct point estimates, incorrect standard errors
res2  # correct point estimates, correct standard errors
plot(res2)
# Log risk ratio per one IQR change in all exposures (not on quantile basis)
dat$x1iqr <- dat$x1/with(dat, diff(quantile(x1, c(.25, .75))))
dat$x2iqr <- dat$x2/with(dat, diff(quantile(x2, c(.25, .75))))
# note that I(x>...) now operates on the untransformed value of x,
# rather than the quantized value
res2 = qgcomp.glm.boot(y ~ z + x1iqr + I(x2iqr>0.1) + I(x2>0.4) + I(x2>0.9),
  family="binomial", expnms = c('x1iqr', 'x2iqr'), data=dat, q=NULL, rr=TRUE, B=200,
  degree=2)
res2
# using parallel processing

qgcomp.glm.boot(y ~ z + x1iqr + I(x2iqr>0.1) + I(x2>0.4) + I(x2>0.9),
  family="binomial", expnms = c('x1iqr', 'x2iqr'), data=dat, q=NULL, rr=TRUE, B=200,
  degree=2, parallel=TRUE, parplan=TRUE)


# weighted model
N=5000
dat4 <- data.frame(id=seq_len(N), x1=runif(N), x2=runif(N), z=runif(N))
dat4$y <- with(dat4, rnorm(N, x1*z + z, 1))
dat4$w=runif(N) + dat4$z*5
qdata = quantize(dat4, expnms = c("x1", "x2"), q=4)$data
# first equivalent models with no covariates
qgcomp.glm.noboot(f=y ~ x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian())
qgcomp.glm.noboot(f=y ~ x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian(),
              weights=w)

set.seed(13)
qgcomp.glm.boot(f=y ~ x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian(),
            weights=w)
# using the correct model
set.seed(13)
qgcomp.glm.boot(f=y ~ x1*z + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian(),
            weights=w, id="id")
(qgcfit <- qgcomp.glm.boot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4,
                       family=gaussian(), weights=w))
```

```
qgcfit$fit
summary(glm(y ~ z + x1 + x2, data = qdata, weights=w))

## End(Not run)
```

---

qgcomp.glm.ee *Quantile g-computation for continuous and binary outcomes*

Description

This function estimates a dose-response parameter representing a one quantile increase in a set of exposures of interest. This model estimates the parameters of a marginal structural model (MSM) based on g-computation with quantized exposures. Note: this function allows non-linear and non-additive effects of individual components of the exposure, as well as non-linear joint effects of the mixture via polynomial basis functions, which increase the computational computational burden due to the need for non-parametric bootstrapping.

Estimating equation methodology is used as the underlying estimation scheme. This allows that observations can be correlated, and is fundementally identical to some implementations of "generalized estimating equations" or GEE. Thus, it allows for a more general set of longitudinal data structures than does the qgcomp.glm.noboot function, because it allows that the outcome can vary over time within an individual. Interpretation of parameters is similar to that of a GEE: this function yields population average estimates of the effect of exposure over time.

Note: GEE (and this function, by extension) does not automatically address all problems of longitudinal data, such as lag structures. Those are up to the investigator to specify correctly.

Note: qgcomp.ee is an equivalent function (slated for deprecation)

Usage

```
qgcomp.glm.ee(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  offset = NULL,
  alpha = 0.05,
  rr = TRUE,
  degree = 1,
  seed = NULL,
  includeX = TRUE,
  verbose = TRUE,
  ...
)
```

```
qgcomp.ee(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  offset = NULL,
  alpha = 0.05,
  rr = TRUE,
  degree = 1,
  seed = NULL,
  includeX = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

f	R style formula
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster). Note that qgcomp.glm.noboot will not produce cluster-appropriate standard errors. qgcomp.glm.ee can be used for this, which will use bootstrap sampling of clusters/individuals to estimate cluster-appropriate standard errors via bootstrapping.
weights	NULL or "case weights" - sampling weights representing a proportional representation in the analysis data
offset	Not yet implemented glm or bayesglm
alpha	alpha level for confidence limit calculation
rr	logical: if using binary outcome and rr=TRUE, qgcomp.glm.ee will estimate risk ratio rather than odds ratio
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic (default = 1).
seed	integer or NULL: random number seed for replicable bootstrap results

includeX	(logical) should the design/predictor matrix be included in the output via the fit and msmfit parameters?
verbose	(logical) give extra messages about fits
...	arguments to glm (e.g. family)

Details

Estimates correspond to the average expected change in the (log) outcome per quantile increase in the joint exposure to all exposures in 'expnms'. Test statistics and confidence intervals are based on a cluster-robust variance that is available in estimating equation methods, which are sometimes referred to as "M-estimators." The robust standard error is then used to estimate Wald-type confidence intervals. Note that no error is assumed for estimated quantiles of exposure, so these are treated as fixed quantities

There is an argument to "family" that is optional, but it defaults to gaussian. Current options allow "gaussian" (or gaussian()), "poisson" or "binomial." This function defaults to canonical links (binomial -> logistic link, gaussian -> identity link). However, setting family="binomial" and rr=TRUE uses a marginal structural model with a log-link (and an underlying conditional model with a logistic link). This mimics the behavior of qgcomp.glm.boot which maximizes backwards compatibility but also is a useful default to avoid convergence issues when using the log-link for conditional models. See examples below.

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the marginal structural model (msmfit) used to estimate the final effect estimates.

See Also

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boot(), qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
set.seed(30)
# continuous outcome
dat <- data.frame(y=rnorm(100), x1=runif(100), x2=runif(100), z=runif(100),
  f = as.factor(sample(c(1,2,3), size=100, replace=TRUE)))
# Conditional linear slope
qgcomp.glm.noboot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4, family=gaussian())
# Marginal linear slope (population average slope, for a purely linear,
#  additive model this will equal the conditional)
qgcomp.glm.ee(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4,
  family=gaussian())
qgcomp.glm.ee(f=y ~ z + x1 + factor(x2) + f, expnms = c('x1', 'x2'), data=dat, q=4,
  family=gaussian())
qgcomp.glm.ee(f=y ~ x1 + x2 + I(x1*x2) + z, expnms = c('x1', 'x2'), data=dat, q=4,
  family=gaussian())
```

```
# no intercept model
qgcomp.glm.ee(f=y ~ -1+z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4,
  family=gaussian())

 # Note that these give different answers! In the first, the estimate is conditional on Z,
 # but in the second, Z is marginalized over via standardization. The estimates
 # can be made approximately the same by centering Z (for linear models), but
 # the conditional estimate will typically have lower standard errors.
 dat$z = dat$z - mean(dat$z)

# Conditional linear slope
qgcomp.glm.noboot(y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4, family=gaussian())
# Marginal linear slope (population average slope, for a purely linear,
#  additive model this will equal the conditional)

qgcomp.glm.ee(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=4,
  family=gaussian())
 ## Not run:

# Population average mixture slope which accounts for non-linearity and interactions
# Note this is one use case for the estimating equations approach: replacing bootstrapping
qgcomp.glm.ee(y ~ z + x1 + x2 + I(x1^2) + I(x2*x1), family="gaussian",
 expnms = c('x1', 'x2'), data=dat, q=4)
qgcomp.glm.boot(y ~ z + x1 + x2 + I(x1^2) + I(x2*x1), family="gaussian",
 expnms = c('x1', 'x2'), data=dat, q=4, B=1000)

# generally non-linear/non-addiive underlying models lead to non-linear mixture slopes
dat$y = dat$y + dat$x1*dat$x2
qgcomp.glm.ee(y ~ z + x1 + x2 + I(x1^2) + I(x2*x1), family="gaussian",
 expnms = c('x1', 'x2'), data=dat, q=4, deg=2)
qgcomp.glm.boot(y ~ z + x1 + x2 + I(x1^2) + I(x2*x1), family="gaussian",
 expnms = c('x1', 'x2'), data=dat, q=4, deg=2, B=1000)

# binary outcome
dat <- data.frame(y=rbinom(50,1,0.5), x1=runif(50), x2=runif(50), z=runif(50))

# Conditional mixture OR
qgcomp.glm.noboot(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
  data=dat, q=2)

#Marginal mixture OR (population average OR - in general, this will not equal the
# conditional mixture OR due to non-collapsibility of the OR)
qgcomp.glm.boot(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
  data=dat, q=2, B=300, MCsize=5000, rr=FALSE)
qgcomp.glm.ee(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
  data=dat, q=2, rr=FALSE)

# Population average mixture RR
qgcomp.glm.boot(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
  data=dat, q=2, B=300, MCsize=5000, rr=TRUE)
qgcomp.glm.ee(y ~ z + x1 + x2, family="binomial", expnms = c('x1', 'x2'),
  data=dat, q=2, rr=TRUE)
# getting the RR with the poisson trick
```

```
qgcomp.glm.ee(y ~ z + x1 + x2, family="poisson", expnms = c('x1', 'x2'),
  data=dat, q=2)
qgcomp.glm.boot(y ~ z + x1 + x2, family="poisson", expnms = c('x1', 'x2'),
  data=dat, q=2, B=300, MCsize=5000)

# Population average mixture RR, indicator variable representation of x2
# note that I(x==...) operates on the quantile-based category of x,
# rather than the raw value
res = qgcomp.glm.ee(y ~ z + x1 + I(x2==1) + I(x2==2) + I(x2==3),
  family="binomial", expnms = c('x1', 'x2'), data=dat, q=4, rr=TRUE)
res$fit
plot(res)

# now add in a non-linear MSM
res2a = qgcomp.glm.boot(y ~ z + x1 + I(x2==1) + I(x2==2) + I(x2==3),
  family="binomial", expnms = c('x1', 'x2'), data=dat, q=4, rr=TRUE,
  degree=2)
res2 = qgcomp.glm.ee(y ~ z + x1 + I(x2==1) + I(x2==2) + I(x2==3),
  family="binomial", expnms = c('x1', 'x2'), data=dat, q=4, rr=TRUE,
  degree=2)

# conditional model estimates (identical point estimates and similar std. errors)
summary(res2a$fit)$coefficients
res2$fit
# msm estimates (identical point estimates and different std. errors)
summary(res2a$msmfit)$coefficients  # correct point estimates, incorrect standard errors
res2  # correct point estimates, correct standard errors
plot(res2)
# Log risk ratio per one IQR change in all exposures (not on quantile basis)
dat$x1iqr <- dat$x1/with(dat, diff(quantile(x1, c(.25, .75))))
dat$x2iqr <- dat$x2/with(dat, diff(quantile(x2, c(.25, .75))))
# note that I(x>...) now operates on the untransformed value of x,
# rather than the quantized value
res2 = qgcomp.glm.ee(y ~ z + x1iqr + I(x2iqr>0.1) + I(x2>0.4) + I(x2>0.9),
  family="binomial", expnms = c('x1iqr', 'x2iqr'), data=dat, q=NULL, rr=TRUE,
  degree=2)
res2


# weighted model
N=5000
dat4 <- data.frame(id=seq_len(N), x1=runif(N), x2=runif(N), z=runif(N))
dat4$y <- with(dat4, rnorm(N, x1*z + z, 1))
dat4$w=runif(N) + dat4$z*5
qdata = quantize(dat4, expnms = c("x1", "x2"), q=4)$data
# first equivalent models with no covariates
qgcomp.glm.noboot(f=y ~ x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian())
qgcomp.glm.noboot(f=y ~ x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian(),
              weights=w)

set.seed(13)
qgcomp.glm.ee(f=y ~ x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian())
qgcomp.glm.ee(f=y ~ x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian(),
```

```
            weights=w)
# using the correct model
set.seed(13)
qgcomp.glm.ee(f=y ~ x1*z + x2, expnms = c('x1', 'x2'), data=dat4, q=4, family=gaussian(),
            weights=w, id="id")
(qgcfit <- qgcomp.glm.ee(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4,
                         family=gaussian(), weights=w))
qgcfit$fit
summary(glm(y ~ z + x1 + x2, data = qdata, weights=w))

## End(Not run)
```

`qgcomp.glm.noboot` *Quantile g-computation for continuous, binary, and count outcomes under linearity/additivity*

Description

This function estimates a linear dose-response parameter representing a one quantile increase in a set of exposures of interest. This function is limited to linear and additive effects of individual components of the exposure. This model estimates the parameters of a marginal structural model (MSM) based on g-computation with quantized exposures. Note: this function is valid only under linear and additive effects of individual components of the exposure, but when these hold the model can be fit with very little computational burden. qgcomp.noboot is an equivalent function (slated for deprecation)

Usage

```
qgcomp.glm.noboot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  bayes = FALSE,
  ...
)

qgcomp.noboot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
```

```
  weights,
  alpha = 0.05,
  bayes = FALSE,
  ...
)
```

Arguments

f	R style formula
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster). Note that qgcomp.glm.noboot will not produce cluster-appropriate standard errors (this parameter is essentially ignored in qgcomp.glm.noboot). qgcomp.glm.boot can be used for this, which will use bootstrap sampling of clusters/individuals to estimate cluster-appropriate standard errors via bootstrapping.
weights	"case weights" - passed to the "weight" argument of glm or bayesglm
alpha	alpha level for confidence limit calculation
bayes	use underlying Bayesian model (arm package defaults). Results in penalized parameter estimation that can help with very highly correlated exposures. Note: this does not lead to fully Bayesian inference in general, so results should be interpreted as frequentist.
...	arguments to glm (e.g. family)

Details

For continuous outcomes, under a linear model with no interaction terms, this is equivalent to g-computation of the effect of increasing every exposure by 1 quantile. For binary/count outcomes outcomes, this yields a conditional log odds/rate ratio(s) representing the change in the expected conditional odds/rate (conditional on covariates) from increasing every exposure by 1 quantile. In general, the latter quantity is not equivalent to g-computation estimates. Hypothesis test statistics and confidence intervals are based on using the delta estimate variance of a linear combination of random variables.

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the weights/standardized coefficients in the positive (pos.weights) and negative (neg.weights) directions.

See Also

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boo qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
set.seed(50)
# linear model
dat <- data.frame(y=runif(50,-1,1), x1=runif(50), x2=runif(50), z=runif(50))
qgcomp.glm.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=gaussian())
# not intercept model
qgcomp.glm.noboot(f=y ~-1+ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=gaussian())
# logistic model
dat2 <- data.frame(y=rbinom(50, 1,0.5), x1=runif(50), x2=runif(50), z=runif(50))
qgcomp.glm.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat2, q=2, family=binomial())
# poisson model
dat3 <- data.frame(y=rpois(50, .5), x1=runif(50), x2=runif(50), z=runif(50))
qgcomp.glm.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat3, q=2, family=poisson())
# weighted model
N=5000
dat4 <- data.frame(y=runif(N), x1=runif(N), x2=runif(N), z=runif(N))
dat4$w=runif(N)*2
qdata = quantize(dat4, expnms = c("x1", "x2"))$data
(qgcfit <- qgcomp.glm.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat4, q=4,
                         family=gaussian(), weights=w))
qgcfit$fit
glm(y ~ z + x1 + x2, data = qdata, weights=w)
```

---

qgcomp.hurdle.boot *Quantile g-computation for hurdle count outcomes*

Description

This function estimates a linear dose-response parameter representing a one quantile increase in a set of exposures of interest for hurdle count outcomes. This function is limited to linear and additive effects of individual components of the exposure. This model estimates the parameters of a marginal structural hurdle count model (MSM) based on g-computation with quantized exposures. Note: this function allows linear and non-additive effects of individual components of the exposure, as well as non-linear joint effects of the mixture via polynomial basis functions, which increase the computational computational burden due to the need for non-parametric bootstrapping.

Usage

```
qgcomp.hurdle.boot(
  f,
  data,
  expnms = NULL,
```

```
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  B = 200,
  degree = 1,
  seed = NULL,
  bayes = FALSE,
  parallel = FALSE,
  MCsize = 10000,
  msmcontrol = hurdlemsm_fit.control(),
  parplan = FALSE,
  ...
)
```

Arguments

f	R style formula
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	"case weights" - passed to the "weight" argument of hurdle. NOTE - this does not work with parallel=TRUE!
alpha	alpha level for confidence limit calculation
B	integer: number of bootstrap iterations (this should typically be >=200, though it is set lower in examples to improve run-time).
degree	polynomial basis function for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic.
seed	integer or NULL: random number seed for replicable bootstrap results
bayes	not currently implemented.
parallel	use (safe) parallel processing from the future and future.apply packages
MCsize	integer: sample size for simulation to approximate marginal zero inflated model parameters. This can be left small for testing, but should be as large as needed to reduce simulation error to an acceptable magnitude (can compare psi coefficients for linear fits with qgcomp.hurdle.noboot to gain some intuition for the level of expected simulation error at a given value of MCsize)

msmcontrol	named list from hurdlemsm_fit.control
parplan	(logical, default=FALSE) automatically set future::plan to plan(multisession) (and set to existing plan, if any, after bootstrapping)
...	arguments to glm (e.g. family)

Details

Hurdle count models allow excess zeros in standard count outcome (e.g. Poisson distributed outcomes). Such models have two components: 1) the probability of arising from a degenerate distribution at zero (versus arising from a count distribution) and 2) the rate parameter of a (possibly truncated > 0) count distribution. Thus, one has the option of allowing exposure and covariate effects on the zero distribution, the count distribution, or both. The zero distribution parameters correspond to log-odds ratios for the probability of arising from the zero distribution. Count distribution parameters correspond to log-rate-ratio parameters. Test statistics and confidence intervals are based on a non-parametric bootstrap, using the standard deviation of the bootstrap estimates to estimate the standard error. The bootstrap standard error is then used to estimate Wald-type confidence intervals. Note that no bootstrapping is done on estimated quantiles of exposure, so these are treated as fixed quantities.

Of note, this function yields marginal estimates of the expected outcome under values of the joint exposure quantiles (e.g. the expected outcome if all exposures are below the 1st quartile). These outcomes can be used to derive estimates of the effect on the marginal expectation of the outcome, irrespective of zero/count portions of the statistical model.

Estimates correspond to the average expected change in the (log) outcome per quantile increase in the joint exposure to all exposures in 'expnms'. Test statistics and confidence intervals are based on a non-parametric bootstrap, using the standard deviation of the bootstrap estimates to estimate the standard error. The bootstrap standard error is then used to estimate Wald-type confidence intervals. Note that no bootstrapping is done on estimated quantiles of exposure, so these are treated as fixed quantities

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the marginal structural model (msmfit) used to estimate the final effect estimates.

See Also

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boot qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
set.seed(50)
n=500
dat <- data.frame(y=rbinom(n, 1, 0.5)*rpois(n, 1.2), x1=runif(n), x2=runif(n), z=runif(n))
# poisson count model, mixture in both portions
## Not run:
# warning: the examples below can take a long time to run
```

```
res = qgcomp.hurdle.boot(f=y ~ x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson", B=1000, MCsize=10000, parallel=TRUE, parplan=TRUE)
qgcomp.hurdle.noboot(f=y ~ x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson")
res

# accuracy for small MCsize is suspect (compare coefficients between boot/noboot versions),
# so re-check with MCsize set to larger value (this takes a long time to run)
res2 = qgcomp.hurdle.boot(f=y ~ x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson", B=1000, MCsize=50000, parallel=TRUE, parplan=TRUE)
 res2
plot(density(res2$bootsamps[4,]))

# negative binomial count model, mixture and covariate in both portions
qgcomp.hurdle.boot(f=y ~ z + x1 + x2 | z + x1 + x2, expnms = c('x1', 'x2'),
   data=dat, q=4, dist="negbin", B=10, MCsize=10000)

# weighted analysis (NOTE THIS DOES NOT WORK WITH parallel=TRUE!)
dat$w = runif(n)*5
qgcomp.hurdle.noboot(f=y ~ z + x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson", weights=w)
# You may see this:
# Warning message:
# In eval(family$initialize) : non-integer #successes in a binomial glm!
qgcomp.hurdle.boot(f=y ~ x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson", B=5, MCsize=50000, parallel=FALSE, weights=w)
# Log rr per one IQR change in all exposures (not on quantile basis)
dat$x1iqr <- dat$x1/with(dat, diff(quantile(x1, c(.25, .75))))
dat$x2iqr <- dat$x2/with(dat, diff(quantile(x2, c(.25, .75))))
# note that I(x>...) now operates on the untransformed value of x,
# rather than the quantized value
res2 = qgcomp.hurdle.boot(f=y ~ z + x1iqr + x2iqr + I(x2iqr>0.1) +
                            I(x2iqr>0.4) + I(x2iqr>0.9) | x1iqr + x2iqr,
                  expnms = c('x1iqr', 'x2iqr'),
                  data=dat, q=NULL, B=2, degree=2, MCsize=2000, dist="poisson")
res2

## End(Not run)
```

---

qgcomp.hurdle.noboot *Quantile g-computation for hurdle count outcomes under linearity/additivity*

Description

This function estimates a linear dose-response parameter representing a one quantile increase in a set of exposures of interest for hurdle count outcomes. This function is limited to linear and additive effects of individual components of the exposure. This model estimates the parameters of a marginal structural hurdle model (MSM) based on g-computation with quantized exposures. Note: this function is valid only under linear and additive effects of individual components of the exposure, but when these hold the model can be fit with very little computational burden.

Usage

```
qgcomp.hurdle.noboot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  bayes = FALSE,
  ...
)
```

Arguments

f	R style formula using syntax from 'pscl' package: depvar ~ indvars_count | indvars_zero
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	"case weights" - passed to the "weight" argument of hurdle.
alpha	alpha level for confidence limit calculation
bayes	not yet implemented
...	arguments to hurdle (e.g. dist)

Details

A hurdle version of quantile g-computation based on the implementation in the 'pscl' package. A hurdle distribution is a mixture distribution in which one of the distributions is a point mass at zero (with probability given by a logistic model), and the other distribution is a discrete or continuous distribution. This estimates the effect of a joint increase in all exposures on 1) the odds of belonging to the "zero" vs. "count" portions of the distribution and/or 2) the rate parameter for the "count" portion of the distribution.

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the weights/standardized coefficients in the positive (pos.weights) and negative (neg.weights) directions.

See Also

qgcomp.hurdle.boot,qgcomp.glm.noboot, qgcomp.cox.noboot, and hurdle

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.multinomial.boot(), qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
set.seed(50)
n=100
dat <- data.frame(y=rbinom(n, 1, 0.5)*rpois(n, 1.2), x1=runif(n), x2=runif(n), z=runif(n))

# poisson count model, mixture in both portions
qgcomp.hurdle.noboot(f=y ~ z + x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=2, dist="poisson")

# negative binomial count model, mixture and covariate in both portions
qgcomp.hurdle.noboot(f=y ~ z + x1 + x2 | z + x1 + x2, expnms = c('x1', 'x2'),
   data=dat, q=2, dist="negbin")
qgcomp.hurdle.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'),
   data=dat, q=2, dist="negbin") # equivalent

# negative binomial count model, mixture only in the 'count' portion of the model
qgcomp.hurdle.noboot(f=y ~ z + x1 + x2 | z, expnms = c('x1', 'x2'), data=dat, q=2, dist="negbin")

# weighted analysis
dat$w = runif(n)*5
qgcomp.hurdle.noboot(f=y ~ z + x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=2, dist="poisson", weights=w)
# Expect this:
# Warning message:
# In eval(family$initialize) : non-integer #successes in a binomial glm!
```

---

qgcomp.multinomial.boot

Quantile g-computation for multinomial outcomes

Description

This function estimates a dose-response parameter representing a one quantile increase in a set of exposures of interest. This model estimates the parameters of a marginal structural model (MSM) based on g-computation with quantized exposures. Note: this function allows linear and non-additive effects of individual components of the exposure, as well as non-linear joint effects of the mixture via polynomial basis functions, which increase the computational computational burden due to the need for non-parametric bootstrapping.

Usage

```
qgcomp.multinomial.boot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  B = 200,
  rr = TRUE,
  degree = 1,
  seed = NULL,
  bayes = FALSE,
  MCsize = nrow(data),
  parallel = FALSE,
  parplan = FALSE,
  ...
)
```

Arguments

f	R style formula
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster). Note that qgcomp.multinomial.noboot will not produce cluster-appropriate standard errors. qgcomp.multinomial.boot can be used for this, which will use bootstrap

	sampling of clusters/individuals to estimate cluster-appropriate standard errors via bootstrapping.
weights	"case weights" - passed to the "weight" argument of multinom
alpha	alpha level for confidence limit calculation
B	integer: number of bootstrap iterations (this should typically be >=200, though it is set lower in examples to improve run-time).
rr	(not used)
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic (default = 1).
seed	integer or NULL: random number seed for replicable bootstrap results
bayes	use underlying Bayesian model (arm package defaults). Results in penalized parameter estimation that can help with very highly correlated exposures. Note: this does not lead to fully Bayesian inference in general, so results should be interpreted as frequentist.
MCsize	integer: sample size for simulation to approximate marginal zero inflated model parameters. This can be left small for testing, but should be as large as needed to reduce simulation error to an acceptable magnitude (can compare psi coefficients for linear fits with qgcomp.multinomial.noboot to gain some intuition for the level of expected simulation error at a given value of MCsize). This likely won't matter much in linear models, but may be important with binary or count outcomes.
parallel	use (safe) parallel processing from the future and future.apply packages
parplan	(logical, default=FALSE) automatically set future::plan to plan(multisession) (and set to existing plan, if any, after bootstrapping)
...	arguments to nnet::multinom

Details

Estimates correspond to the average expected change in the probability of an outcome type per quantile increase in the joint exposure to all exposures in 'expnms'. Test statistics and confidence intervals are based on a non-parametric bootstrap, using the standard deviation of the bootstrap estimates to estimate the standard error. The bootstrap standard error is then used to estimate Wald-type confidence intervals. Note that no bootstrapping is done on estimated quantiles of exposure, so these are treated as fixed quantities

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the marginal structural model (msmfit) used to estimate the final effect estimates.

See Also

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
data("metals") # from qgcomp package
# create categorical outcome from the existing continuous outcome (usually, one will already exist)
metals$ycat = factor(quantize(metals, "y",q=4)$data$y, levels=c("0", "1", "2", "3"),
                     labels=c("cct", "ccg", "aat", "aag"))
# restrict to smaller dataset for simplicity
smallmetals = metals[,c("ycat", "arsenic", "lead", "cadmium", "mage35")]

### 1: Define mixture and underlying model ####
mixture = c("arsenic", "lead", "cadmium")
f0 = ycat ~ arsenic + lead + cadmium # the multinomial model
# (be sure that factor variables are properly coded ahead of time in the dataset)
rr = qgcomp.multinomial.boot(
 f0,
 expnms = mixture,
 q=4,
 data = smallmetals,
 B = 5, # set to higher values in real examples
 MCsize = 100,  # set to higher values in small samples
 )

rr2 = qgcomp.multinomial.noboot(
 f0,
 expnms = mixture,
 q=4,
 data = smallmetals
 )

 ### 5: Create summary qgcomp object for nice printing ####

 summary(rr, tests=c("H")) # include homogeneity test

 # 95% confidence intervals
 #confint(rr, level=0.95)
 #rr$breaks # quantile cutpoints for exposures
 # homogeneity_test(rr)
 #joint_test(rr)

qdat = simdata_quantized(
  outcometype="multinomial",
  n=10000, corr=c(-.9), coef=cbind(c(.2,-.2,0,0), c(.1,.1,.1,.1)),
  q = 4
)

 rr_sim = qgcomp.multinomial.noboot(
  y~x1+x2+x3+x4,
  expnms = c("x1", "x2", "x3", "x4"),
  q=4,
  data = qdat
 )

 rr_sim2 = qgcomp.multinomial.boot(
```

```
  y~x1+x2+x3+x4,
  expnms = c("x1", "x2", "x3", "x4"),
  q=4,
  data = qdat,
  B=1
 )
```

`qgcomp.multinomial.noboot` *Quantile g-computation for multinomial outcomes*

Description

Quantile g-computation for multinomial outcomes

Usage

```
qgcomp.multinomial.noboot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  bayes = FALSE,
  ...
)
```

Arguments

`f`	R style formula
`data`	data frame
`expnms`	character vector of exposures of interest
`q`	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
`breaks`	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
`id`	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster). Note that qgcomp.multinomial.noboot will not produce cluster-appropriate standard

	errors (this parameter is essentially ignored in qgcomp.multinomial.noboot). qgcomp.qgcomp.multinomial.boot can be used for this, which will use bootstrap sampling of clusters/individuals to estimate cluster-appropriate standard errors via bootstrapping.
weights	"case weights" - passed to the "weight" argument of multinom
alpha	alpha level for confidence limit calculation
bayes	Logical, Not yet implemented (gives and error if set to TRUE)
...	arguments to nnet::multinom

Value

a qgcompmultfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the weights/standardized coefficients in the positive and negative directions (weights).

See Also

qgcomp.glm.noboot, multinom

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boot(), qgcomp.partials(), qgcomp.zi.boot(), qgcomp.zi.noboot()

Examples

```
data("metals") # from qgcomp package
# create categorical outcome from the existing continuous outcome (usually, one will already exist)
metals$ycat = factor(quantize(metals, "y",q=4)$data$y, levels=c("0", "1", "2", "3"),
                     labels=c("cct", "ccg", "aat", "aag"))
# restrict to smaller dataset for simplicity
smallmetals = metals[,c("ycat", "arsenic", "lead", "cadmium", "mage35")]

### 1: Define mixture and underlying model ####
mixture = c("arsenic", "lead", "cadmium")
f0 = ycat ~ arsenic + lead + cadmium # the multinomial model
# (be sure that factor variables are properly coded ahead of time in the dataset)

rr = qgcomp.multinomial.noboot(
 f0,
 expnms = mixture,
 q=4,
 data = smallmetals,
 )

 ### 5: Create summary qgcomp object for nice printing ####

 summary(rr, tests=c("H")) # include homogeneity test

 # 95% confidence intervals
 confint(rr, level=0.95)
 rr$breaks # quantile cutpoints for exposures
```

```
# homogeneity_test(rr)
joint_test(rr)
```

---

qgcomp.partials *Partial effect sizes, confidence intervals, hypothesis tests*

Description

Obtain effect estimates for "partial positive" and "partial negative" effects using quantile g-computation. This approach uses sample splitting to evaluate the overall impact of a set of variables with effects in a single direction, where, using training data, all variables with effects in the same direction are grouped.

Usage

```
qgcomp.partials(
  fun = c("qgcomp.glm.noboot", "qgcomp.cox.noboot", "qgcomp.zi.noboot"),
  traindata = NULL,
  validdata = NULL,
  expnms = NULL,
  .fixbreaks = TRUE,
  .globalbreaks = FALSE,
  ...
)
```

Arguments

fun	character variable in the set "qgcomp.glm.noboot" (binary, count, continuous outcomes), "qgcomp.cox.noboot" (survival outcomes), "qgcomp.zi.noboot" (zero inflated outcomes). This describes which qgcomp package function is used to fit the model. (default = "qgcomp.glm.noboot")
traindata	Data frame with training data
validdata	Data frame with validation data
expnms	Exposure mixture of interest
.fixbreaks	(logical, overridden by .globalbreaks) Use the same quantile cutpoints in the training and validation data (selected in the training data). As of version 2.8.11, the default is TRUE, whereas it was implicitly FALSE in prior verions. Setting to TRUE increases variance but greatly decreases bias in smaller samples.
.globalbreaks	(logical, if TRUE, overrides .fixbreaks) Use the same quantile cutpoints in the training and validation data (selected in combined training and validation data). As of version 2.8.11, the default is TRUE, whereas it was implicitly FALSE in prior verions. Setting to TRUE increases variance but greatly decreases bias in smaller samples.
...	Arguments to qgcomp.glm.noboot, qgcomp.cox.noboot, or qgcomp.zi.noboot

Details

In the basic (non bootstrapped) qgcomp functions, the positive and negative "sums of coefficients" or "partial effect sizes" are given, which equal the sum of the negative and positive coefficients in the underlying model. Unfortunately, these partial effects don't constitute variables for which we can derive confidence intervals or hypothesis tests, so they are mainly for exploratory purposes. By employing sample splitting, however, we can obtain better estimates of these partial effects.

Sample splitting proceeds by partitioning the data into two samples (40/60 training/validtion split seems acceptable in many circumstances). The "overall mixture effect" is then estimated in the training data, and the mixture variables with positive and negative coefficients are split into separate groups. These two different groups are then used as "the mixture of interest" in two additional qgcomp fits, where the mixture of interest is adjusted for the other exposure variables. For example, if the "positive partial effect" is of interest, then this effect is equal to the sum of the coefficients in the qgcomp model fit to the validation data, with the mixture of interest selected by the original fit to the training data (note that some of these coefficients may be negative in the fit to the validation data - this is expected and necessary for valid hypothesis tests).

The positive/negative partial effects are necessarily exploratory, but sample splitting preserves the statistical properties at the expense of wider confidence intervals and larger variances. The two resulting mixture groups groups should be inspected for

Value

A 'qgcompmultifit' object, which inherits from `list`, which contains

posmix character vector of variable names with positive coefficients in the qgcomp model fit to the training data

negmix character vector of variable names with negative coefficients in the qgcomp model fit to the training data

pos.fit a qgcompfit object fit to the validation data, in which the exposures of interest are contained in 'posmix'

neg.fit a qgcompfit object fit to the validation data, in which the exposures of interest are contained in 'negmix'

See Also

Other qgcomp_methods: `qgcomp.cch.noboot()`, `qgcomp.cox.boot()`, `qgcomp.cox.noboot()`, `qgcomp.glm.boot()`, `qgcomp.glm.ee()`, `qgcomp.glm.noboot()`, `qgcomp.hurdle.boot()`, `qgcomp.hurdle.noboot()`, `qgcomp.multinomial.boot()`, `qgcomp.multinomial.noboot()`, `qgcomp.zi.boot()`, `qgcomp.zi.noboot()`

Examples

```
set.seed(123223)
dat = qgcomp::simdata_quantized(n=1000, outcomtype="continuous", cor=c(.75, 0),
                                b0=0, coef=c(0.25,-0.25,0,0), q=4)
cor(dat)
# overall fit (more or less null due to counteracting exposures)
(overall <- qgcomp.glm.noboot(f=y~., q=NULL, expnms=c("x1", "x2", "x3", "x4"), data=dat))

# partial effects using 40% training/60% validation split
```

```
trainidx <- sample(1:nrow(dat), round(nrow(dat)*0.4))
valididx <- setdiff(1:nrow(dat),trainidx)
traindata = dat[trainidx,]
validdata = dat[valididx,]
splitres <- qgcomp.partials(fun="qgcomp.glm.noboot", f=y~., q=NULL,
    traindata=traindata,validdata=validdata, expnms=c("x1", "x2", "x3", "x4"))
splitres
## Not run:
# under the null, both should give null results
set.seed(123223)
dat = simdata_quantized(n=1000, outcomtype="continuous", cor=c(.75, 0),
                                b0=0, coef=c(0,0,0,0), q=4)
# 40% training/60% validation
trainidx2 <- sample(1:nrow(dat), round(nrow(dat)*0.4))
valididx2 <- setdiff(1:nrow(dat),trainidx2)
traindata2 <- dat[trainidx2,]
validdata2 <- dat[valididx2,]
splitres2 <- qgcomp.partials(fun="qgcomp.glm.noboot", f=y~.,
   q=NULL, traindata=traindata2,validdata=validdata2, expnms=c("x1", "x2", "x3", "x4"))
splitres2

# 60% training/40% validation
trainidx3 <- sample(1:nrow(dat), round(nrow(dat)*0.6))
valididx3 <- setdiff(1:nrow(dat),trainidx3)
traindata3 <- dat[trainidx3,]
validdata3 <- dat[valididx3,]
splitres3 <- qgcomp.partials(fun="qgcomp.glm.noboot", f=y~., q=NULL,
    traindata=traindata3,validdata=validdata3, expnms=c("x1", "x2", "x3", "x4"))
splitres3

# survival outcome
set.seed(50)
N=1000
dat = simdata_quantized(n=1000, outcomtype="survival", cor=c(.75, 0, 0, 0, 1),
                                b0=0, coef=c(1,0,0,0,0,1), q=4)
names(dat)[which(names(dat=="x5"))] = "z"
trainidx4 <- sample(1:nrow(dat), round(nrow(dat)*0.6))
valididx4 <- setdiff(1:nrow(dat),trainidx4)
traindata4 <- dat[trainidx4,]
validdata4 <- dat[valididx4,]
expnms=paste0("x", 1:5)
f = survival::Surv(time, d)~x1 + x2 + x3 + x4 + x5 + z
(fit1 <- survival::coxph(f, data = dat))
(overall <- qgcomp.cox.noboot(f, expnms = expnms, data = dat))
(splitres4 <- qgcomp.partials(fun="qgcomp.cox.noboot", f=f, q=4,
               traindata=traindata4,validdata=validdata4,
                expnms=expnms))

 # zero inflated count outcome
 set.seed(50)
 n=1000
 dat <- data.frame(y= (yany <- rbinom(n, 1, 0.5))*(ycnt <- rpois(n, 1.2)), x1=runif(n)+ycnt*0.2,
                      x2=runif(n)-ycnt*0.2, x3=runif(n),
```

```
                    x4=runif(n) , z=runif(n))
 # poisson count model, mixture in both portions, but note that the qgcomp.partials
 # function defines the "positive" variables only  by the count portion of the model
 (overall5 <- qgcomp.zi.noboot(f=y ~ z + x1 + x2 + x3 + x4 | x1 + x2 + x3 + x4 + z,
                 expnms = c("x1", "x2", "x3", "x4"),
                  data=dat, q=4, dist="poisson"))

trainidx5 <- sample(1:nrow(dat), round(nrow(dat)*0.6))
valididx5 <- setdiff(1:nrow(dat),trainidx5)
traindata5 <- dat[trainidx5,]
validdata5 <- dat[valididx5,]
splitres5 <- qgcomp.partials(fun="qgcomp.zi.noboot",
    f=y ~ x1 + x2 + x3 + x4 + z | x1 + x2 + x3 + x4 + z, q=4,
    traindata=traindata5, validdata=validdata5,
    expnms=c("x1", "x2", "x3", "x4"))
splitres5


## End(Not run)
```

`qgcomp.survcurve.boot` *Survival curve data from a qgcomp survival fit*

Description

It is often of interest to examine survival curves from qgcomp.cox.boot models. They can be useful for checking assumptions about how well the marginal structural model conforms to the underlying conditional model, such that the overall fit approximates the non-linearity in the underlying model. This function will yield survival curves, but no measures of uncertainty.

Usage

```
qgcomp.survcurve.boot(x, ...)
```

Arguments

`x`	a `qgcompfit` object from `qgcomp.cox.boot`
`...`	not used

Value

a list of data.frames:

mdfpop Average Survival curve (survival, time) based on marginal structural model, averaged over the population at every quantile of exposure

cdfpop Population average survival curve (survival, time) based on the underlying conditional model

mdfq Survival curves (survival, time) for each quantile based on marginal structural model

cdfq Survival curves (survival, time) for each quantile based on underlying conditional model

Examples

```
set.seed(50)
N=200
dat <- data.frame(time=(tmg <- pmin(.1,rweibull(N, 10, 0.1))),
                  d=1.0*(tmg<0.1), x1=runif(N), x2=runif(N), z=runif(N))
expnms=paste0("x", 1:2)
f = survival::Surv(time, d)~x1 + x2
(fit1 <- survival::coxph(f, data = dat))
(obj <- qgcomp.cox.noboot(f, expnms = expnms, data = dat))
## Not run:
(obj2 <- qgcomp.cox.boot(f, expnms = expnms, data = dat, B=10, MCsize=20000))
curves = cox.survcurve.boot(obj2)
rbind(head(curves$mdfq),tail(curves$mdfq))

## End(Not run)
```

---

qgcomp.tobit.noboot *Quantile g-computation for left-censored outcomes*

Description

This function yields population average effect estimates for (possibly left censored) outcomes #'

Usage

```
qgcomp.tobit.noboot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights = NULL,
  cluster = NULL,
  alpha = 0.05,
  left = -Inf,
  right = Inf,
  ...
)
```

Arguments

f	an r formula representing the conditional model for the outcome, given all exposures and covariates. Interaction terms that include exposure variables should be represented via the AsIs function
data	data frame

expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	Not used
weights	"case weights" - passed to the "weight" argument of AER::tobit. Note this must be a standalone vector so that it might accept, for example, an argument like data$weights but will not accept "weights" if there is a variable "weights" in the dataset representing the weights. tobit
cluster	Passed to AER::tobit (Optional variable that identifies groups of subjects, used in computing the robust variance. Like model variables, this is searched for in the dataset pointed to by the data argument). Note: this will result in robust standard errors being returned unless robust=FALSE is used in the function call.
alpha	alpha level for confidence limit calculation
left	Passed to AER::tobit (From tobit docs: left limit for the censored dependent variable y. If set to -Inf, y is assumed not to be left-censored.)
right	Passed to AER::tobit (From tobit docs: right limit for the censored dependent variable y. If set to Inf, the default, y is assumed not to be right-censored.)
...	arguments to AER::tobit (e.g. dist), and consequently to survival::survreg

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the weights/standardized coefficients in the positive (pos.weights) and negative (neg.weights) directions.

Examples

```
# First, a demonstration that the tobit model will return identical results
# (with variation due to numerical algorithms)
# in the case of no censored values
set.seed(50)
# linear model
dat <- data.frame(y=runif(50,-1,1), x1=runif(50), x2=runif(50), z=runif(50))
qgcomp.glm.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2, family=gaussian())
qgcomp.tobit.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'), data=dat, q=2)
# not intercept model
qgcomp.glm.noboot(f=y ~-1+ z + x1 + x2, expnms = c('x1', 'x2'), data=dat,
    q=2, family=gaussian())
qgcomp.tobit.noboot(f=y ~-1+ z + x1 + x2, expnms = c('x1', 'x2'), data=dat,
    q=2, dist="gaussian", left = -Inf, right = Inf)
# Next, a demonstration that the tobit model address censoring
```

```
uncens_dat = qgcomp:::.dgm_quantized(N=1000, b0=0, coef=c(.33,.33,.33,.0))
uncens_dat$ycens = ifelse(uncens_dat$y>0, uncens_dat$y, 0) # censor at zero
uncens_dat$ycens1 = ifelse(uncens_dat$y>0, uncens_dat$y, 1) # censor at higher value
# q set to NULL because the exposures are already quantized
# again, first check the uncensored outcome for agreement
qgcomp.glm.noboot(f= y ~ x1+x2+x3+x4, expnms = c('x1', 'x2', 'x3', 'x4'),
    data=uncens_dat, q=NULL, family=gaussian())
qgcomp.tobit.noboot(f= y ~x1+x2+x3+x4, expnms = c('x1', 'x2', 'x3', 'x4'),
    data=uncens_dat, q=NULL, dist="gaussian", left = -Inf, right = Inf)
# Next, after censoring
ft_std <- qgcomp.glm.noboot(f= ycens ~ x1+x2+x3+x4,
    expnms = c('x1', 'x2', 'x3', 'x4'), data=uncens_dat, q=NULL,
    family=gaussian())
ft_tobit <- qgcomp.tobit.noboot(f= ycens ~x1+x2+x3+x4,
    expnms = c('x1', 'x2', 'x3', 'x4'), data=uncens_dat, q=NULL,
    dist="gaussian", left = 0, right = Inf)
# note the tobit regression will be closer to the estimates given when
# fitting with the uncensored outcome (which will typically not be available)
summary(ft_std)
summary(ft_tobit)
ft_std1 <- qgcomp.glm.noboot(f= ycens1 ~ x1+x2+x3+x4,
    expnms = c('x1', 'x2', 'x3', 'x4'), data=uncens_dat, q=NULL,
    family=gaussian())
ft_tobit1 <- qgcomp.tobit.noboot(f= ycens1 ~x1+x2+x3+x4,
    expnms = c('x1', 'x2', 'x3', 'x4'), data=uncens_dat, q=NULL,
    dist="gaussian", left = 1, right = Inf)
# the bias from standard methods is more extreme at higher censoring levels
summary(ft_std1)
summary(ft_tobit1)
```

`qgcomp.zi.boot` *Quantile g-computation for zero-inflated count outcomes*

Description

This function estimates a linear dose-response parameter representing a one quantile increase in a set of exposures of interest for zero-inflated count outcomes. This function is limited to linear and additive effects of individual components of the exposure. This model estimates the parameters of a marginal structural zero-inflated count model (MSM) based on g-computation with quantized exposures. Note: this function allows linear and non-additive effects of individual components of the exposure, as well as non-linear joint effects of the mixture via polynomial basis functions, which increase the computational computational burden due to the need for non-parametric bootstrapping.

Usage

```
qgcomp.zi.boot(
  f,
  data,
  expnms = NULL,
```

```
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  B = 200,
  degree = 1,
  seed = NULL,
  bayes = FALSE,
  parallel = FALSE,
  MCsize = 10000,
  msmcontrol = zimsm_fit.control(),
  parplan = FALSE,
  ...
)
```

Arguments

f	R style formula
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	"case weights" - passed to the "weight" argument of zeroinfl. NOTE - this does not work with parallel=TRUE!
alpha	alpha level for confidence limit calculation
B	integer: number of bootstrap iterations (this should typically be >=200, though it is set lower in examples to improve run-time).
degree	polynomial basis function for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic.)
seed	integer or NULL: random number seed for replicable bootstrap results
bayes	not currently implemented.
parallel	use (safe) parallel processing from the future and future.apply packages
MCsize	integer: sample size for simulation to approximate marginal zero inflated model parameters. This can be left small for testing, but should be as large as needed to reduce simulation error to an acceptable magnitude (can compare psi coefficients for linear fits with qgcomp.zi.noboot to gain some intuition for the level of expected simulation error at a given value of MCsize)

msmcontrol	named list from zimsm_fit.control
parplan	(logical, default=FALSE) automatically set future::plan to plan(multisession) (and set to existing plan, if any, after bootstrapping)
...	arguments to glm (e.g. family)

Details

Zero-inflated count models allow excess zeros in standard count outcome (e.g. Poisson distributed outcomes). Such models have two components: 1) the probability of arising from a degenerate distribution at zero (versus arising from a count distribution) and 2) the rate parameter of a count distribution. Thus, one has the option of allowing exposure and covariate effects on the zero distribution, the count distribution, or both. The zero distribution parameters correspond to log-odds ratios for the probability of arising from the zero distribution. Count distribution parameters correspond to log-rate-ratio parameters. Test statistics and confidence intervals are based on a non-parametric bootstrap, using the standard deviation of the bootstrap estimates to estimate the standard error. The bootstrap standard error is then used to estimate Wald-type confidence intervals. Note that no bootstrapping is done on estimated quantiles of exposure, so these are treated as fixed quantities.

Of note, this function yields marginal estimates of the expected outcome under values of the joint exposure quantiles (e.g. the expected outcome if all exposures are below the 1st quartile). These outcomes can be used to derive estimates of the effect on the marginal expectation of the outcome, irrespective of zero-inflated/count portions of the statistical model.

Estimates correspond to the average expected change in the (log) outcome per quantile increase in the joint exposure to all exposures in 'expnms'. Test statistics and confidence intervals are based on a non-parametric bootstrap, using the standard deviation of the bootstrap estimates to estimate the standard error. The bootstrap standard error is then used to estimate Wald-type confidence intervals. Note that no bootstrapping is done on estimated quantiles of exposure, so these are treated as fixed quantities

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the marginal structural model (msmfit) used to estimate the final effect estimates.

See Also

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boot(), qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.noboot()

Examples

```
set.seed(50)
n=100
dat <- data.frame(y=rbinom(n, 1, 0.5)*rpois(n, 1.2), x1=runif(n), x2=runif(n), z=runif(n))
# poisson count model, mixture in both portions
## Not run:
# warning: the examples below can take a long time to run
```

```
res = qgcomp.zi.boot(f=y ~ x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson", B=1000, MCsize=10000, parallel=TRUE, parplan=TRUE)
qgcomp.zi.noboot(f=y ~ x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson")
res

# accuracy for small MCsize is suspect (compare coefficients between boot/noboot versions),
# so re-check with MCsize set to larger value (this takes a long time to run)
res2 = qgcomp.zi.boot(f=y ~ x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson", B=1000, MCsize=50000, parallel=TRUE, parplan=TRUE)
 res2
plot(density(res2$bootsamps[4,]))

# negative binomial count model, mixture and covariate in both portions
qgcomp.zi.boot(f=y ~ z + x1 + x2 | z + x1 + x2, expnms = c('x1', 'x2'),
   data=dat, q=4, dist="negbin", B=10, MCsize=10000)

# weighted analysis (NOTE THIS DOES NOT WORK WITH parallel=TRUE!)
dat$w = runif(n)*5
qgcomp.zi.noboot(f=y ~ z + x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson", weights=w)
# Expect this:
# Warning message:
# In eval(family$initialize) : non-integer #successes in a binomial glm!
qgcomp.zi.boot(f=y ~ x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=4, dist="poisson", B=5, MCsize=50000, parallel=FALSE, weights=w)
# Log rr per one IQR change in all exposures (not on quantile basis)
dat$x1iqr <- dat$x1/with(dat, diff(quantile(x1, c(.25, .75))))
dat$x2iqr <- dat$x2/with(dat, diff(quantile(x2, c(.25, .75))))
# note that I(x>...) now operates on the untransformed value of x,
# rather than the quantized value
res2 = qgcomp.zi.boot(y ~ z + x1iqr + x2iqr + I(x2iqr>0.1) + I(x2>0.4) + I(x2>0.9) | x1iqr + x2iqr,
                  family="binomial", expnms = c('x1iqr', 'x2iqr'), data=dat, q=NULL, B=2,
                    degree=2, MCsize=200, dist="poisson")
res2

## End(Not run)
```

---

qgcomp.zi.noboot *Quantile g-computation for zero-inflated count outcomes under linearity/additivity*

Description

This function estimates a linear dose-response parameter representing a one quantile increase in a set of exposures of interest for zero-inflated count outcomes. This function is limited to linear and additive effects of individual components of the exposure. This model estimates the parameters of a marginal structural zero-inflated model (MSM) based on g-computation with quantized exposures. Note: this function is valid only under linear and additive effects of individual components of the exposure, but when these hold the model can be fit with very little computational burden.

Usage

```
qgcomp.zi.noboot(
  f,
  data,
  expnms = NULL,
  q = 4,
  breaks = NULL,
  id = NULL,
  weights,
  alpha = 0.05,
  bayes = FALSE,
  ...
)
```

Arguments

f	R style formula using syntax from 'pscl' package: depvar ~ indvars_count | indvars_zero
data	data frame
expnms	character vector of exposures of interest
q	NULL or number of quantiles used to create quantile indicator variables representing the exposure variables. If NULL, then gcomp proceeds with untransformed version of exposures in the input datasets (useful if data are already transformed, or for performing standard g-computation)
breaks	(optional) NULL, or a list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	"case weights" - passed to the "weight" argument of zeroinfl.
alpha	alpha level for confidence limit calculation
bayes	not yet implemented
...	arguments to zeroinfl (e.g. dist)

Details

A zero-inflated version of quantile g-computation based on the implementation in the 'pscl' package. A zero-inflated distribution is a mixture distribution in which one of the distributions is a point mass at zero (with probability given by a logistic model), and the other distribution is a discrete or continuous distribution. This estimates the effect of a joint increase in all exposures on 1) the odds of belonging to the "zero" vs. "count" portions of the distribution and/or 2) the rate parameter for the "count" portion of the distribution.

Value

a qgcompfit object, which contains information about the effect measure of interest (psi) and associated variance (var.psi), as well as information on the model fit (fit) and information on the weights/standardized coefficients in the positive (pos.weights) and negative (neg.weights) directions.

See Also

Other qgcomp_methods: qgcomp.cch.noboot(), qgcomp.cox.boot(), qgcomp.cox.noboot(), qgcomp.glm.boot(), qgcomp.glm.ee(), qgcomp.glm.noboot(), qgcomp.hurdle.boot(), qgcomp.hurdle.noboot(), qgcomp.multinomial.boot(), qgcomp.multinomial.noboot(), qgcomp.partials(), qgcomp.zi.boot()

Examples

```
set.seed(50)
n=100
dat <- data.frame(y=rbinom(n, 1, 0.5)*rpois(n, 1.2), x1=runif(n), x2=runif(n), z=runif(n))

# poisson count model, mixture in both portions
qgcomp.zi.noboot(f=y ~ z + x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=2, dist="poisson")

# negative binomial count model, mixture and covariate in both portions
qgcomp.zi.noboot(f=y ~ z + x1 + x2 | z + x1 + x2, expnms = c('x1', 'x2'),
   data=dat, q=2, dist="negbin")
qgcomp.zi.noboot(f=y ~ z + x1 + x2, expnms = c('x1', 'x2'),
   data=dat, q=2, dist="negbin") # equivalent

# negative binomial count model, mixture only in the 'count' portion of the model
qgcomp.zi.noboot(f=y ~ z + x1 + x2 | z, expnms = c('x1', 'x2'), data=dat, q=2, dist="negbin")

# weighted analysis
dat$w = runif(n)*5
qgcomp.zi.noboot(f=y ~ z + x1 + x2 | x1 + x2, expnms = c('x1', 'x2'),
    data=dat, q=2, dist="poisson", weights=w)
# Expect this:
# Warning message:
# In eval(family$initialize) : non-integer #successes in a binomial glm!
```

---

quantize *Quantizing exposure data*

Description

Create variables representing indicator functions with cutpoints defined by quantiles. Output a list that includes: 1) a dataset that is a copy of data, except that the variables whose names are included in the expnms variable are transformed to their quantized version and 2) an unnamed list of the quantile cutpoints that are used for each of the variables that were quantized

Usage

```
quantize(data, expnms, q = 4, breaks = NULL)
```

Arguments

data	a data frame
expnms	a character vector with the names of the columns to be quantized
q	integer, number of quantiles used in creating quantized variables
breaks	(optional) list of (equal length) numeric vectors that characterize the minimum value of each category for which to break up the variables named in expnms. This is an alternative to using 'q' to define cutpoints.

Details

This function creates categorical variables in place of the exposure variables named in 'expnms.' For example, a continuous exposure 'x1' will be replaced in the output data by another 'x1' that takes on values 0:(q-1), where, for example, the value 1 indicates that the original x1 value falls between the first and the second quantile.

Value

A list containing the following fields

data a quantized version of the original dataframe

breaks a list of the quantile cutpoints used to create the quantized variables which includes a very small number for the minimum and a very large number for the maximum to avoid causing issues when using these breaks to quantize new data.

Examples

```
set.seed(1232)
dat = data.frame(y=runif(100), x1=runif(100), x2=runif(100), z=runif(100))
qdata = quantize(data=dat, expnms=c("x1", "x2"), q=4)
table(qdata$data$x1)
table(qdata$data$x2)
summary(dat[c("y", "z")]);summary(qdata$data[c("y", "z")]) # not touched
dat = data.frame(y=runif(100), x1=runif(100), x2=runif(100), z=runif(100))
# using 'breaks' requires specifying min and max (the qth quantile)
# example with theoretical quartiles (could be other relevant values)
qdata2 = quantize(data=dat, expnms=c("x1", "x2"),
   breaks=list(c(-1e64, .25, .5, .75, 1e64),
               c(-1e64, .25, .5, .75, 1e64)
               ))
table(qdata2$data$x1)
table(qdata2$data$x2)
```

se_comb *Calculate standard error of weighted linear combination of random variables*

Description

This function uses the Delta method to calculate standard errors of linear functions of variables (similar to `lincom` in Stata). Generally, users will not need to call this function directly.

Usage

```
se_comb(expnms, covmat, grad = NULL)
```

Arguments

`expnms`	a character vector with the names of the columns to be of interest in the covariance matrix for a which a standard error will be calculated (e.g. same as expnms in qgcomp fit)
`covmat`	covariance matrix for parameters, e.g. from a model or bootstrap procedure
`grad`	the "weight" vector for calculating the contribution of each variable in expnms to the final standard error. For a linear combination, this is equal to a vector of ones (and is set automatically). Or can be calculated via the grad_poly procedure, in the case of coming up with proper weights when the combination of expnms derives from a polynomial function (as in qgcomp.glm.boot with degree>1).

Details

This function takes inputs of a set of exposure names (character vector) and a covariance matrix (with colnames/rownames that contain the full set of exposure names), as well as a possible `grad` parameter to calculate the variance of a weighted combination of the exposures in `expnms`, where the weights are based off of `grad` (which defaults to 1, so that this function yields the variance of a sum of all variables in `expnms`)

Here is simple version of the delta method for a linear combination of three model coefficients:

$f(\beta) = \beta_1 + \beta_2 + \beta_3$ given gradient vector

$$G = [d(f(\beta))/d\beta_1 = 1, d(f(\beta))/d\beta_2 = 1, d(f(\beta))/d\beta_3 = 1]$$

$t(G)Cov(\beta)G$ = delta method variance, where t() is the transpose operator

Examples

```
vcov = rbind(c(1.2, .9),c(.9, 2.0))
colnames(vcov) <- rownames(vcov) <- expnms <- c("x1", "x2")
se_comb(expnms, vcov, c(1, 0))^2 # returns the given variance
se_comb(expnms, vcov, c(1, 1)) # default linear MSM fit: all exposures
# have equal weight
se_comb(expnms, vcov, c(.3, .1)) # used when one exposure contributes
  # to the overall fit more than others  = d(msmeffect)/dx
```

`simdata_quantized` *Simulate quantized exposures for testing methods*

Description

Simulate quantized exposures for testing methods

Usage

```
simdata_quantized(
  outcometype = c("continuous", "logistic", "survival", "multinomial"),
  n = 100,
  corr = NULL,
  b0 = 0,
  coef = c(1, 0, 0, 0),
  q = 4,
  yscale = 1,
  shape0 = 3,
  scale0 = 5,
  censtime = 4,
  ncheck = TRUE,
  ...
)
```

Arguments

`outcometype`	Character variable that is one of c("continuous", "logistic", "survival"). Selects what type of outcome should be simulated (or how). continuous = normal, continous outcome, logistic= binary outcome from logistic model, survival = right censored survival outcome from Weibull model.
`n`	Sample size
`corr`	NULL, or vector of correlations between the first exposure and subsequent exposures (if length(corr) < (length(coef)-1), then this will be backfilled with zeros)
`b0`	(continuous, binary outcomes) model intercept
`coef`	Vector of coefficients for the outcome (i.e. model coefficients for exposures). The length of this determines the number of exposures.
`q`	Number of levels or "quanta" of each exposure
`yscale`	(continuous outcomes) error scale (residual error) for normally distributed outcomes
`shape0`	(survival outcomes) baseline shape of weibull distribution rweibull
`scale0`	(survival outcomes) baseline scale of weibull distribution rweibull
`censtime`	(survival outcomes) administrative censoring time
`ncheck`	(logical, default=TRUE) adjust sample size if needed so that exposures are exactly evenly distributed (so that qgcomp::quantize(exposure) = exposure)
`...`	unused

Details

Simulate continuous (normally distributed errors), binary (logistic function), or event-time outcomes as a linear function

Value

a data frame

See Also

qgcomp.glm.boot, and qgcomp.glm.noboot

Examples

```
set.seed(50)
qdat = simdata_quantized(
  outcometype="continuous",
  n=10000, corr=c(.9,.3), coef=c(1,1,0,0),
  q = 8
)
cor(qdat)
qdat = simdata_quantized(
  outcometype="continuous",
  n=10000, corr=c(-.9,.3), coef=c(1,2,0,0),
  q = 4
)
cor(qdat)
table(qdat$x1)
qgcomp.glm.noboot(y~.,data=qdat)

qdat = simdata_quantized(
  outcometype="multinomial",
  n=10000, corr=c(-.9), coef=cbind(c(1,-1,0,0), c(1,.2,0,0)),
  q = 4
)
```

---

split_data *Perform sample splitting*

Description

This is a convenience function to split the input data into two independent sets, possibly accounting for single level clustering. These two sets can be used with qgcomp.partials to get "partial" positive/negative effect estimates from the original data, where sample splitting is necessary to get valid confidence intervals and p-values. Sample splitting is also useful for any sort of exploratory model selection, where the training data can be used to select the model and the validation model used to generate the final estimates (this process should not be iterative - e.g. no "checking" the results in the validation data and then re-fitting, as this invalidates inference in the validation set.)

E.g. you could use the training data to select non-linear terms for the model and then re-fit in validation data to get unbiased estimates.

Usage

```
split_data(data, cluster = NULL, prop.train = 0.4)
```

Arguments

data	A data.frame for use in qgcomp fitting
cluster	NULL (default) or character value naming a cluster identifier in the data. This is to prevent observations from a single cluster being in both the training and validation data, which reduces the effectiveness of sample splitting.
prop.train	proportion of the original dataset (or proportion of the clusters identified via the 'cluster' parameter) that are used in the training data (default=0.4)

Value

A list of the following type: list(trainidx = trainidx, valididx = valididx, traindata = traindata, validdata = validdata)

e.g. if you call spl = split_data(dat), then spl$traindata will contain a 40% sample from the original data, spl$validdata will contain the other 60% and spl$trainidx, spl$valididx will contain integer indexes that track the row numbers (from the original data dat) that have the training and validation samples.

Examples

```
data(metals)
set.seed(1231124)
spl = split_data(metals)
Xnm <- c(
  'arsenic','barium','cadmium','calcium','chromium','copper',
 'iron','lead','magnesium','manganese','mercury','selenium','silver',
 'sodium','zinc'
)
dim(spl$traindata) # 181 observations = 40% of total
dim(spl$validdata) # 271 observations = 60% of total
splitres <- qgcomp.partials(fun="qgcomp.glm.noboot", f=y~., q=4,
  traindata=spl$traindata,validdata=spl$validdata, expnms=Xnm)
splitres

# also used to compare linear vs. non-linear fits (useful if you have enough data)
set.seed(1231)
spl = split_data(metals, prop.train=.5)
lin = qgcomp.glm.boot(f=y~., q=4, expnms=Xnm, B=5, data=spl$traindata)
nlin1 = qgcomp.glm.boot(f=y~. + I(manganese^2) + I(calcium^2), expnms=Xnm, deg=2,
  q=4, B=5, data=spl$traindata)
nlin2 = qgcomp.glm.boot(f=y~. + I(arsenic^2) + I(cadmium^2), expnms=Xnm, deg=2,
  q=4, B=5, data=spl$traindata)
AIC(lin);AIC(nlin1);AIC(nlin2)
```

```
# linear has lowest training AIC, so base final fit off that (and bootstrap not needed)
qgcomp.glm.noboot(f=y~., q=4, expnms=Xnm, data=spl$validdata)
```

`summary.qgcompmultfit` *Summarize gcompmultfit object*

Description

Summary printing to include coefficients, standard errors, hypothesis tests, weights

Usage

```
## S3 method for class 'qgcompmultfit'
summary(object, ..., tests = NULL)
```

Arguments

`object`	Result from qgcomp multinomial fit (qgcompmultfit object).
`...`	Unused
`tests`	Character vector (e.g. c("global", "homogeneity")) that determine the types of hypothesis tests that are printed

Value

qgcompmulttest object (list) with results of a chi-squared test

---

`tidy.qgcompfit` *Tidy method for qgcompfit object*

Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies cross models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return. (Description taken from `tidyr::tidy` help file.)

Usage

```
## S3 method for class 'qgcompfit'
tidy(x, conf.level = 1 - x$alpha, exponentiate = FALSE, quick = FALSE, ...)
```

Arguments

x	a agcompfit object created by qgcomp().
conf.level	Real number between 0 and 1 corresponding to nominal percentage/100 of confidence limit (e.g. conf.level=0.95 means 95 per cent confidence intervals). Defaults to 1-alpha level of qgcompfit.
exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
quick	Logical indiciating if the only the term and estimate columns should be returned. Often useful to avoid time consuming covariance and standard error calculations. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. Cautionary note: Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass conf.lvel = 0.9, all computation will proceed using conf.level = 0.95. Additionally, if you pass newdata = my_tibble to an augment() method that does not accept a newdata argument, it will use the default value for the data argument.

---

vc_comb *Calculate covariance matrix between one random variable and a linear combination of random variables*

Description

This function uses the Delta method to calculate a covariance matrix of linear functions of variables and is used internally in qgcomp. Generally, users will not need to call this function directly.

Usage

```
vc_comb(aname = "(Intercept)", expnms, covmat, grad = NULL)
```

Arguments

aname	character scalar with the name of the first column of interest (e.g. variable A in the examples given in the details section)
expnms	a character vector with the names of the columns to be of interest in the covariance matrix for a which a standard error will be calculated (e.g. same as expnms in qgcomp fit)
covmat	covariance matrix for parameters, e.g. from a model or bootstrap procedure
grad	not yet used

Details

This function takes inputs of a name of random variable (character), as set of exposure names (character vector) and a covariance matrix (with colnames/rownames that contain the indepdendent variable and the full set of exposure names). See se_comb for details on variances of sums of random variables. Briefly, for variables A, B and C with covariance matrix Cov(A,B,C), we can calculate the covariance Cov(A,B+C) with the formula Cov(A,B) + Cov(A,C), and Cov(A,B+C+D) = Cov(A,B) + Cov(A,C) + Cov(A,D), and so on.

Value

A covariance matrix

Examples

```
vcov = rbind(c(0.010051348, -0.0039332248, -0.0036965571),
             c(-0.003933225,  0.0051807876,  0.0007706792),
             c(-0.003696557,  0.0007706792,  0.0050996587))
colnames(vcov) <- rownames(vcov) <- c("(Intercept)", "x1", "x2")
expnms <- rownames(vcov)[2:3]
aname = rownames(vcov)[1]
vc_comb(aname, expnms, vcov) # returns the given covariance matrix
```

---

zimsm_fit *Secondary prediction method for the (zero-inflated) qgcomp MSM.*

Description

this is an internal function called by qgcomp.zi.boot, but is documented here for clarity. Generally, users will not need to call this function directly.

Usage

```
zimsm_fit(
  f,
  qdata,
  intvals,
  expnms,
  main = TRUE,
  degree = 1,
  id = NULL,
  weights,
  MCsize = 10000,
  containmix = list(count = TRUE, zero = TRUE),
  bayes = FALSE,
  x = FALSE,
  msmcontrol = zimsm_fit.control(),
  ...
)
```

Arguments

f	an r formula representing the conditional model for the outcome, given all exposures and covariates. Interaction terms that include exposure variables should be represented via the AsIs function
qdata	a data frame with quantized exposures (as well as outcome and other covariates)
intvals	sequence, the sequence of integer values that the joint exposure is 'set' to for estimating the msm. For quantile g-computation, this is just 0:(q-1), where q is the number of quantiles of exposure.
expnms	a character vector with the names of the columns in qdata that represent the exposures of interest (main terms only!)
main	logical, internal use: produce estimates of exposure effect (psi) and expected outcomes under g-computation and the MSM
degree	polynomial bases for marginal model (e.g. degree = 2 allows that the relationship between the whole exposure mixture and the outcome is quadratic. Default=1)
id	(optional) NULL, or variable name indexing individual units of observation (only needed if analyzing data with multiple observations per id/cluster)
weights	not yet implemented
MCsize	integer: sample size for simulation to approximate marginal hazards ratios
containmix	named list of logical scalars with names "count" and "zero"
bayes	not used
x	keep design matrix? (logical)
msmcontrol	named list from zimsm_fit.control
...	arguments to zeroinfl (e.g. dist)

Details

This function first computes expected outcomes under hypothetical interventions to simultaneously set all exposures to a specific quantile. These predictions are based on g-computation, where the exposures are 'quantized', meaning that they take on ordered integer values according to their ranks, and the integer values are determined by the number of quantile cutpoints used. The function then takes these expected outcomes and fits an additional model (a marginal structural model) with the expected outcomes as the outcome and the intervention value of the exposures (the quantile integer) as the exposure. Under causal identification assumptions and correct model specification, the MSM yields a causal exposure-response representing the incremental change in the expected outcome given a joint intervention on all exposures.

See Also

qgcomp.cox.boot, and qgcomp.cox.noboot

Examples

```
set.seed(50)
n=100
## Not run:
dat <- data.frame(y=rbinom(n, 1, 0.5)*rpois(n, 1.2), x1=runif(n), x2=runif(n), z=runif(n))
expnms = c("x1", "x2")
q = 4
qdata = quantize(dat, q=q, expnms=expnms)$data
f = y ~ x1 + x2 + z | 1
msmfit <- zimsm_fit(f, qdata, intvals=(1:q)-1, expnms, main=TRUE,
  degree=1, id=NULL, MCsize=10000, containmix=list(count=TRUE, zero=FALSE),
  x=FALSE)
msmfit$msmfit

## End(Not run)
```

---

zimsm_fit.control *Control of fitting parameters for zero inflated MSMs*

Description

this is an internal function called by qgcomp.zi.boot, but is documented here for clarity. Generally, users will not need to call this function directly.

Usage

```
zimsm_fit.control(predmethod = c("components", "catprobs"))
```

Arguments

predmethod	character in c("components", "catprobs"). "components" simulates from the model parameters directly while "catprobs" simulates outcomes from the category specific probabilities, which is output from predict.zeroinfl. The former is slightly more flexible and stable, but the latter is preferred in zero inflated negative bionomial models.

Details

Provides fine control over zero inflated MSM fitting

Index