Package 'lava'

June 30, 2026

Type Package

Title Latent Variable Models

Version 1.9.2

Author Klaus K. Holst [aut, cre],
Benedikt Sommer [ctb],
Brice Ozenne [ctb],
Thomas Gerds [ctb]

Maintainer Klaus K. Holst <klaus@holst.it>

Description A general implementation of Structural Equation Models with latent variables (MLE, 2SLS, and composite likelihood estimators) with both continuous, censored, and ordinal outcomes (Holst and Budtz-Joergensen (2013) <doi:10.1007/s00180-012-0344-y>). Mixture latent variable models and non-linear latent variable models (Holst and Budtz-Joergensen (2020) <doi:10.1093/biostatistics/kxy082>). The package also provides methods for graph exploration (d-separation, back-door criterion), simulation of general non-linear latent variable models, and estimation of influence functions for a broad range of statistical models.

URL https://kkholst.github.io/lava/, https://github.com/kkholst/lava

BugReports https://github.com/kkholst/lava/issues

License Apache License (== 2.0)

LazyLoad yes

Depends R (>= 3.5.0)

Imports cli, future.apply, graphics, grDevices, methods, numDeriv, progressr, stats, survival, SQUAREM, utils

Suggests KernSmooth, Rgraphviz, data.table, ellipse, fields, future, geepack, graph, knitr, rmarkdown, igraph (>= 0.6), lavaSearch2, lme4 (>= 1.1.35.1), MASS, Matrix (>= 1.6.3), mets (>= 1.1), nlme, optimx, polycor, quantreg, rgl, svglite, targeted (>= 0.4), testthat (>= 0.11), vdiffr, visNetwork

VignetteBuilder knitr,rmarkdown

ByteCompile yes

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

Repository CRAN

Date/Publication 2026-06-30 16:40:11 UTC

Contents

addvar . . . 5
backdoor . . . 5
baptize . . . 6
binomial.rd . . . 6
blockdiag . . . 8
bmd . . . 8
bmidata . . . 9
bootstrap . . . 9
bootstrap.lvm . . . 10
brisa . . . 11
By . . . 12
c.estimate . . . 12
c.summary.estimate . . . 13
calcium . . . 14
cancel . . . 14
children . . . 15
click . . . 15
closed_testing . . . 17
Col . . . 18
colorbar . . . 19
Combine . . . 20
commutation . . . 21
compare . . . 21
complik . . . 22
confband . . . 23
confint.lvmfit . . . 26
confpred . . . 27
constrain<- . . . 28
contr . . . 31
correlation . . . 32
covariance . . . 33
csplit . . . 35
curly . . . 36
deprdiag . . . 37
devcoords . . . 37
diagtest . . . 38
dsep.lvm . . . 39

equivalence . . . 40
estimate.array . . . 41
estimate.default . . . 41
estimate.formula . . . 48
estimate.glm . . . 49
estimate.list . . . 49
estimate.lvm . . . 50
eventTime . . . 52
Expand . . . 55
fplot . . . 56
getSAS . . . 57
gof . . . 58
Graph . . . 60
Grep . . . 60
hubble . . . 61
hubble2 . . . 62
IC.default . . . 62
iid . . . 63
images . . . 63
index . . . 65
index.lvm . . . 66
indoorenv . . . 67
information . . . 67
intercept . . . 68
intervention.lvm . . . 69
Inverse . . . 70
ksmooth2 . . . 71
labels<- . . . 72
lava.options . . . 73
lvm . . . 74
makemissing . . . 75
measurement.error . . . 76
merge.estimate . . . 77
Missing . . . 78
missingdata . . . 79
mixture . . . 80
Model . . . 82
modelsearch . . . 83
multinomial . . . 84
mvnmix . . . 85
na.pass0 . . . 87
NA2x . . . 87
nldata . . . 88
NR . . . 88
nsem . . . 90
ordinal<- . . . 90
ordreg . . . 91
parpos . . . 92

partialcor . . . 92
path . . . 93
pcor . . . 94
PD . . . 95
pdfconvert . . . 96
plot.estimate . . . 97
plot.lvm . . . 98
plot.sim . . . 100
plotConf . . . 103
predict.lvm . . . 105
predictlvm . . . 107
predict_glm . . . 108
Print . . . 109
Range.lvm . . . 109
rbind.Surv . . . 110
regression<- . . . 111
revdiag . . . 113
rmvar . . . 113
rotate2 . . . 114
scheffe . . . 115
score . . . 115
semdata . . . 116
serotonin . . . 116
sim.default . . . 117
sim.lvm . . . 119
spaghetti . . . 124
stack.estimate . . . 126
subset.lvm . . . 128
summary.estimate . . . 129
summary.sim . . . 130
timedep . . . 131
toformula . . . 132
tr . . . 133
trim . . . 134
twindata . . . 134
twostage . . . 135
twostage.lvmfit . . . 135
twostageCV . . . 138
vars . . . 139
var_ic . . . 141
vec . . . 141
wait . . . 142
wkm . . . 142
wrapvec . . . 143
zibreg . . . 144
%++% . . . 145
%ni% . . . 146

Index **148**

addvar *Add variable to (model) object*

Description

Generic method for adding variables to model object

Usage

```
addvar(x, ...)
```

Arguments

x	Model object
...	Additional arguments

Author(s)

Klaus K. Holst

backdoor *Backdoor criterion*

Description

Check backdoor criterion of a lvm object

Usage

```
backdoor(object, f, cond, ..., return.graph = FALSE)
```

Arguments

object	lvm object
f	formula. Conditioning, z, set can be given as y~x|z
cond	Vector of variables to conditon on
...	Additional arguments to lower level functions
return.graph	Return moral ancestral graph with z and effects from x removed

Examples

```
m <- lvm(y~c2,c2~c1,x~c1,m1~x,y~m1, v1~c3, x~c3,v1~y,
         x~z1, z2~z1, z2~z3, y~z3+z2+g1+g2+g3)
ll <- backdoor(m, y~x)
backdoor(m, y~x|c1+z1+g1)
```

baptize *Label elements of object*

Description

Generic method for labeling elements of an object

Usage

```
baptize(x, ...)
```

Arguments

x	Object
...	Additional arguments

Author(s)

Klaus K. Holst

binomial.rd *Define constant risk difference or relative risk association for binary exposure*

Description

Set up model as defined in Richardson, Robins and Wang (2017).

Usage

```
binomial.rd(
  x,
  response,
  exposure,
  target.model,
  nuisance.model,
  exposure.model = binomial.lvm(),
  ...
)
```

Arguments

x	model
response	response variable (character or formula)
exposure	exposure variable (character or formula)
target.model	variable defining the linear predictor for the target model
nuisance.model	variable defining the linear predictor for the nuisance model
exposure.model	model for exposure (default binomial logit link)
...	additional arguments to lower level functions

Examples

```
## ------------------------------------------------------------
## binomial.rd: constant risk-difference model
##   P(Y=1|Z=1) - P(Y=1|Z=0) = tanh(lp)
## ------------------------------------------------------------
m <- lvm()
regression(m) <- z ~ x
regression(m) <- lp ~ x
regression(m) <- op ~ x
intercept(m, ~lp) <- 0.4   ## constant linear predictor for RD
intercept(m, ~op) <- 0     ## odds product = exp(0) = 1
distribution(m, ~lp) <- normal.lvm(sd = 0)
distribution(m, ~op) <- normal.lvm(sd = 0)
m <- binomial.rd(m, response = "y", exposure = "z",
                 target.model = "lp", nuisance.model = "op")
set.seed(1)
d <- sim(m, n = 2000)
## Empirical risk difference should be close to tanh(0.4)
mean(d$y[d$z == 1]) - mean(d$y[d$z == 0])
tanh(0.4)

## Formula interface: response ~ exposure | target | nuisance
m2 <- lvm()
regression(m2) <- z ~ x
regression(m2) <- lp ~ x
regression(m2) <- op ~ x
m2 <- binomial.rd(m2, y ~ z | lp | op)

## ------------------------------------------------------------
## binomial.rr: constant relative-risk model
##   log(P(Y=1|Z=1) / P(Y=1|Z=0)) = lp
## ------------------------------------------------------------
m <- lvm()
regression(m) <- z ~ x
regression(m) <- lp ~ x
regression(m) <- op ~ x
intercept(m, ~lp) <- log(1.5)   ## constant log relative-risk
intercept(m, ~op) <- 0          ## odds product = 1
distribution(m, ~lp) <- normal.lvm(sd = 0)
distribution(m, ~op) <- normal.lvm(sd = 0)
```

```
m <- binomial.rr(m, response = "y", exposure = "z",
                 target.model = "lp", nuisance.model = "op")
set.seed(1)
d <- sim(m, n = 2000)
## Empirical log-RR should be close to log(1.5)
log(mean(d$y[d$z == 1]) / mean(d$y[d$z == 0]))
log(1.5)
```

---

blockdiag *Combine matrices to block diagonal structure*

Description

Combine matrices to block diagonal structure

Usage

```
blockdiag(x, ..., pad = 0)
```

Arguments

x	matrix
...	additional matrices
pad	value outside block-diagonal

Author(s)

Klaus K. Holst

Examples

```
A <- diag(3)+1
blockdiag(A,A,A,pad=NA)
```

---

bmd *Longitudinal Bone Mineral Density Data (Wide format)*

Description

Bone Mineral Density Data consisting of 112 girls randomized to receive calcium og placebo. Longitudinal measurements of bone mineral density (g/cm^2) measured approximately every 6th month in 3 years.

Format

data.frame

Source

Vonesh & Chinchilli (1997), Table 5.4.1 on page 228.

See Also

calcium

`bmidata` *Data*

Description

Description

Format

data.frame

`bootstrap` *Generic bootstrap method*

Description

Generic method for calculating bootstrap statistics

Usage

```
bootstrap(x, ...)
```

Arguments

`x`	Model object
`...`	Additional arguments

Author(s)

Klaus K. Holst

See Also

`bootstrap.lvm` `bootstrap.lvmfit`

`bootstrap.lvm` *Calculate bootstrap estimates of a lvm object*

Description

Draws non-parametric bootstrap samples

Usage

```
## S3 method for class 'lvm'
bootstrap(x,R=100,data,fun=NULL,control=list(),
                        p, parametric=FALSE, bollenstine=FALSE,
                        constraints=TRUE,sd=FALSE, mc.cores,
                        future.args=list(future.seed=TRUE),
                        ...)

## S3 method for class 'lvmfit'
bootstrap(x,R=100,data=model.frame(x),
                           control=list(start=coef(x)),
                           p=coef(x), parametric=FALSE, bollenstine=FALSE,
                           estimator=x$estimator,weights=Weights(x),...)
```

Arguments

x	lvm-object.
R	Number of bootstrap samples
data	The data to resample from
fun	Optional function of the (bootstrapped) model-fit defining the statistic of interest
control	Options to the optimization routine
p	Parameter vector of the null model for the parametric bootstrap
parametric	If TRUE a parametric bootstrap is calculated. If FALSE a non-parametric (row-sampling) bootstrap is computed.
bollenstine	Bollen-Stine transformation (non-parametric bootstrap) for bootstrap hypothesis testing.
constraints	Logical indicating whether non-linear parameter constraints should be included in the bootstrap procedure
sd	Logical indicating whether standard error estimates should be included in the bootstrap procedure
mc.cores	Optional number of cores for parallel computing. If omitted future.apply will be used (see future::plan)
future.args	arguments to future.apply::future_lapply
...	Additional arguments, e.g. choice of estimator.
estimator	String definining estimator, e.g. 'gaussian' (see estimator)
weights	Optional weights matrix used by estimator

Value

A `bootstrap.lvm` object.

Author(s)

Klaus K. Holst

See Also

`confint.lvmfit`

Examples

```
m <- lvm(y~x)
d <- sim(m,100)
e <- estimate(lvm(y~x), data=d)
 ## Reduce Ex.Timings
B <- bootstrap(e,R=50,mc.cores=1)
B
```

---

brisa *Simulated data*

Description

Simulated data

Format

data.frame

Source

Simulated

By *Apply a Function to a Data Frame Split by Factors*

Description

Apply a Function to a Data Frame Split by Factors

Usage

```
By(x, INDICES, FUN, COLUMNS, array = FALSE, ...)
```

Arguments

x	Data frame
INDICES	Indices (vector or list of indices, vector of column names, or formula of column names)
FUN	A function to be applied to data frame subsets of 'data'.
COLUMNS	(Optional) subset of columns of x to work on
array	if TRUE an array/matrix is always returned
...	Additional arguments to lower-level functions

Details

Simple wrapper of the 'by' function

Author(s)

Klaus K. Holst

Examples

```
By(datasets::CO2,~Treatment+Type,colMeans,~conc)
By(datasets::CO2,~Treatment+Type,colMeans,~conc+uptake)
```

c.estimate *Concatenate estimate objects*

Description

When all arguments are `estimate` objects, they are merged into a single `estimate` object. When some arguments are not `estimate` objects but are named numeric scalars/vectors, the merged `estimate` object is returned with an `"extra"` attribute containing the provided values. This is useful for bundling auxiliary per-iteration information (e.g., convergence status) alongside an estimate for use with `sim.default()`.

Usage

```
## S3 method for class 'estimate'
c(..., as.list = FALSE)
```

Arguments

...	estimate objects and/or named numeric values
as.list	if TRUE the returned object will be of class list and not an estimate object.

Details

arguments drop.ic, paired, sep are passed to merge.estimate

Value

An estimate object. If extra named numeric values were provided, the result carries an "extra" attribute (a named numeric vector).

See Also

sim.default() merge.estimate()

Examples

```
e <- estimate(coef = c(a = 1, b = 2), vcov = diag(2) * 0.1)
# Bundle estimate with extra information
res <- c(e, converged = 1, niter = 10)
attr(res, "extra")
```

---

c.summary.estimate *Concatenate summary.estimate objects*

Description

When all arguments are summary.estimate objects, they are merged into a single summary.estimate object. When some arguments are not summary.estimate objects but are named numeric scalars/vectors, an object of class summary.estimate is returned with an additional extra attribute that contains the provided numeric scalars/vectors. This is useful for bundling auxiliary per-iteration information (e.g., convergence status) alongside an estimate for use with sim.default().

Usage

```
## S3 method for class 'summary.estimate'
c(...)
```

Arguments

...	summary.estimate objects and/or named numeric values

Examples

```
e1 <- estimate(coef = 1, IC = scale(rnorm(10)), id = 1:10, labels = "a1")
e2 <- estimate(coef = 2, IC = scale(rnorm(10)), id = 1:10, labels = "a2")

# concatenating two summary.estimate objects
c(e1, e2)

# concatenating one summary.estimate object with one numerical variable
ss <- c(summary(e1), niter = 2)
print(ss)
attributes(ss)$extra
```

`calcium` *Longitudinal Bone Mineral Density Data*

Description

Bone Mineral Density Data consisting of 112 girls randomized to receive calcium og placebo. Longitudinal measurements of bone mineral density (g/cm^2) measured approximately every 6th month in 3 years.

Format

A data.frame containing 560 (incomplete) observations. The 'person' column defines the individual girls of the study with measurements at visiting times 'visit', and age in years 'age' at the time of visit. The bone mineral density variable is 'bmd' (g/cm^2).

Source

Vonesh & Chinchilli (1997), Table 5.4.1 on page 228.

`cancel` *Generic cancel method*

Description

Generic cancel method

Usage

```
cancel(x, ...)
```

Arguments

`x`	Object
`...`	Additioal arguments

Author(s)

Klaus K. Holst

`children` *Extract children or parent elements of object*

Description

Generic method for memberships from object (e.g. a graph)

Usage

```
children(object, ...)
```

Arguments

`object`	Object
`...`	Additional arguments

Author(s)

Klaus K. Holst

`click` *Identify points on plot*

Description

Extension of the `identify` function

Usage

```
## Default S3 method:
click(x, y=NULL, label=TRUE, n=length(x), pch=19, col="orange", cex=3, ...)
idplot(x, y ,..., id=list(), return.data=FALSE)
```

Arguments

x	X coordinates
...	Additional arguments parsed to plot function
y	Y coordinates
label	Should labels be added?
n	Max number of inputs to expect
pch	Symbol
col	Colour
cex	Size
id	List of arguments parsed to click function
return.data	Boolean indicating if selected points should be returned

Details

For the usual 'X11' device the identification process is terminated by pressing any mouse button other than the first. For the 'quartz' device the process is terminated by pressing either the pop-up menu equivalent (usually second mouse button or 'Ctrl'-click) or the 'ESC' key.

Author(s)

Klaus K. Holst

See Also

idplot, identify

Examples

```
if (interactive()) {
    n <- 10; x <- seq(n); y <- runif(n)
    plot(y ~ x); click(x,y)

    data(iris)
    l <- lm(Sepal.Length ~ Sepal.Width*Species,iris)
    res <- plotConf(l,var2="Species")## ylim=c(6,8), xlim=c(2.5,3.3))
    with(res, click(x,y))

    with(iris, idplot(Sepal.Length,Petal.Length))
}
```

`closed_testing` *Closed testing procedure*

Description

Given k hypotheses $H_1, \ldots, H_k$ all $2^k - 1$ intersection hypotheses are calculated and adjusted p-values are obtained for H_j is calculated as the max p-value of all intersection hypotheses containing H_j. Example, for $k = 3$, the adjusted p-value for H_1 will be obtained from $\{(H_1, H_2, H_3), (H_1, H_2), (H_1, H_3), (H_1)\}$.

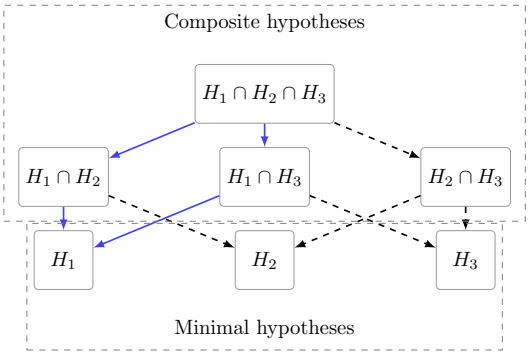
Usage

```
closed_testing(object, test = test_wald, ...)
```

Arguments

`object`	`estimate` object
`test`	function that conducts hypothesis test. See details below.
`...`	Additional arguments passed to `test`

Details



The function `test` should be a function `function(object, index, ...)` which as its first argument takes an `estimate` object and with an argument `index` which is a integer vector specifying which subcomponents of `object` to test. The ellipsis argument can be any other arguments used in the test function. The function `test_wald()` is an example of valid test function (which has an additional argument `null` in reference to the above mentioned ellipsis arguments).

References

Marcus, R; Peritz, E; Gabriel, KR (1976). "On closed testing procedures with special reference to ordered analysis of variance". Biometrika. 63 (3): 655–660.

Examples

```
m <- lvm()
regression(m, c(y1,y2,y3,y4)~x) <- c(0, 0.25, 0, 0.25)
regression(m, to=endogenous(m), from="u") <- 1
variance(m,endogenous(m)) <- 1
set.seed(1)
d <- sim(m, 200)
l1 <- lm(y1~x,d)
l2 <- lm(y2~x,d)
l3 <- lm(y3~x,d)
l4 <- lm(y4~x,d)

(a <- merge(l1, l2, l3, l4, subset=2))
if (requireNamespace("mets",quietly=TRUE)) {
   alpha_zmax(a)
}
adj <- closed_testing(a, test = test_wald, null = 0)
adj
adj$p.value
summary(adj)
```

Col *Generate a transparent RGB color*

Description

This function transforms a standard color (e.g. "red") into an transparent RGB-color (i.e. alpha-blend<1).

Usage

```
Col(col, alpha = 0.2, locate = 0)
```

Arguments

col	color (numeric or character)
alpha	degree of transparency (0-1)
locate	Choose colour (with mouse)

Details

This only works for certain graphics devices (Cairo-X11 (x11 as of R>=2.7), quartz, pdf, ...).

Value

A character vector with elements of 7 or 9 characters, # followed by the red, blue, green and optionally alpha values in hexadecimal (after rescaling to '0 ... 255').

Author(s)

Klaus K. Holst

Examples

```
plot(runif(1000),cex=runif(1000,0,4),
     col=Col(c("darkblue","orange"),0.5),pch=16)
```

`colorbar` *Add color-bar to plot*

Description

Add color-bar to plot

Usage

```
colorbar(
  clut = Col(rev(rainbow(11, start = 0, end = 0.69)), alpha),
  x.range = c(-0.5, 0.5),
  y.range = c(-0.1, 0.1),
  values = seq(clut),
  digits = 2,
  label.offset,
  srt = 45,
  cex = 0.5,
  border = NA,
  alpha = 0.5,
  position = 1,
  direction = c("horizontal", "vertical"),
  ...
)
```

Arguments

`clut`	Color look-up table
`x.range`	x range
`y.range`	y range
`values`	label values
`digits`	number of digits
`label.offset`	label offset
`srt`	rotation of labels
`cex`	text size
`border`	border of color bar rectangles

alpha	Alpha (transparency) level 0-1
position	Label position left/bottom (1) or top/right (2) or no text (0)
direction	horizontal or vertical color bars
...	additional low level arguments (i.e. parsed to text)

Examples

```
## Not run:
plotNeuro(x,roi=R,mm=-18,range=5)
colorbar(clut=Col(rev(rainbow(11,start=0,end=0.69)),0.5),
         x=c(-40,40),y.range=c(84,90),values=c(-5:5))

colorbar(clut=Col(rev(rainbow(11,start=0,end=0.69)),0.5),
         x=c(-10,10),y.range=c(-100,50),values=c(-5:5),
         direction="vertical",border=1)

## End(Not run)
```

Combine *Report estimates across different models*

Description

Report estimates across different models

Usage

```
Combine(x, ...)
```

Arguments

x	list of model objects
...	additional arguments to lower-level functions

Author(s)

Klaus K. Holst

Examples

```
data(serotonin)
m1 <- lm(cau ~ age*gene1 + age*gene2,data=serotonin)
m2 <- lm(cau ~ age + gene1,data=serotonin)
m3 <- lm(cau ~ age*gene2,data=serotonin)

Combine(list(A=m1,B=m2,C=m3),fun=function(x)
     c("_____"="",R2=" "%++%format(summary(x)$r.squared,digits=2)))
```

commutation *Finds the unique commutation matrix*

Description

Finds the unique commutation matrix K: $Kvec(A) = vec(A^t)$

Usage

```
commutation(m, n = m)
```

Arguments

m	rows
n	columns

Author(s)

Klaus K. Holst

compare *Statistical tests*

Description

Performs Likelihood-ratio, Wald and score tests

Usage

```
compare(object, ...)
```

Arguments

object	lvmfit-object
...	Additional arguments to low-level functions

Value

Matrix of test-statistics and p-values

Author(s)

Klaus K. Holst

See Also

modelsearch, equivalence

Examples

```
m <- lvm();
regression(m) <- c(y1,y2,y3) ~ eta; latent(m) <- ~eta
regression(m) <- eta ~ x
m2 <- regression(m, c(y3,eta) ~ x)
set.seed(1)
d <- sim(m,1000)
e <- estimate(m,d)
e2 <- estimate(m2,d)

compare(e)

compare(e,e2) ## LRT, H0: y3<-x=0
compare(e,scoretest=y3~x) ## Score-test, H0: y3~x=0
compare(e2,par=c("y3~x")) ## Wald-test, H0: y3~x=0

B <- diag(2); colnames(B) <- c("y2~eta","y3~eta")
compare(e2,contrast=B,null=c(1,1))

B <- rep(0,length(coef(e2))); B[1:3] <- 1
compare(e2,contrast=B)

compare(e,scoretest=list(y3~x,y2~x))
```

---

complik *Composite Likelihood for probit latent variable models*

Description

Estimate parameters in a probit latent variable model via a composite likelihood decomposition.

Usage

```
complik(
  x,
  data,
  k = 2,
  type = c("all", "nearest"),
  pairlist,
  messages = 0,
  estimator = "normal",
  quick = FALSE,
  ...
)
```

Arguments

x	lvm-object
data	data.frame
k	Size of composite groups
type	Determines number of groups. With type="nearest" (default) only neighboring items will be grouped, e.g. for k=2 (y1,y2),(y2,y3),... With type="all" all combinations of size k are included
pairlist	A list of indices specifying the composite groups. Optional argument which overrides k and type but gives complete flexibility in the specification of the composite likelihood
messages	Control amount of messages printed
estimator	Model (pseudo-likelihood) to use for the pairs/groups
quick	If TRUE the parameter estimates are calculated but all additional information such as standard errors are skipped
...	Additional arguments parsed on to lower-level functions

Value

An object of class estimate.complik inheriting methods from lvm

Author(s)

Klaus K. Holst

See Also

estimate

Examples

```
m <- lvm(c(y1,y2,y3)~b*x+1*u[0],latent=~u)
ordinal(m,K=2) <- ~y1+y2+y3
d <- sim(m,50,seed=1)
if (requireNamespace("mets", quietly=TRUE)) {
   e1 <- complik(m,d,control=list(trace=1),type="all")
}
```

---

confband *Add Confidence limits bar to plot*

Description

Add Confidence limits bar to plot

Usage

```
confband(
  x,
  lower,
  upper,
  center = NULL,
  line = TRUE,
  delta = 0.07,
  centermark = 0.03,
  pch,
  blank = TRUE,
  vert = TRUE,
  polygon = FALSE,
  alpha = 1,
  step = FALSE,
  ...
)
```

Arguments

x	Position (x-coordinate if vert=TRUE, y-coordinate otherwise)
lower	Lower limit (if NULL no limits is added, and only the center is drawn (if not NULL))
upper	Upper limit
center	Center point
line	If FALSE do not add line between upper and lower bound
delta	Length of limit bars
centermark	Length of center bar
pch	Center symbol (if missing a line is drawn)
blank	If TRUE a white ball is plotted before the center is added to the plot
vert	If TRUE a vertical bar is plotted. Otherwise a horizontal bar is used
polygon	If TRUE polygons are added between 'lower' and 'upper'
alpha	transparency of fill-color of polygon (alpha<1)
step	Type of polygon (step-function or piecewise linear)
...	Additional low level arguments (e.g. col, lwd, lty,...)

Author(s)

Klaus K. Holst

See Also

confband

Examples

```
plot(0,0,type="n",xlab="",ylab="")
confband(0.5,-0.5,0.5,0,col="darkblue")
confband(0.8,-0.5,0.5,0,col="darkred",vert=FALSE,pch=1,cex=1.5)

set.seed(1)
K <- 20
est <- rnorm(K)
se <- runif(K,0.2,0.4)
x <- cbind(est,est-2*se,est+2*se,runif(K,0.5,2))
x[c(3:4,10:12),] <- NA
rownames(x) <- unlist(lapply(letters[seq(K)],function(x) paste(rep(x,4),collapse="")))
rownames(x)[which(is.na(est))] <- ""
signif <- sign(x[,2])==sign(x[,3])
forestplot(x,text.right=FALSE)
forestplot(x[,-4],sep=c(2,15),col=signif+1,box1=TRUE,delta=0.2,pch=16,cex=1.5)
forestplot(x,vert=TRUE,text=FALSE)
forestplot(x,vert=TRUE,text=FALSE,pch=NA)
##forestplot(x,vert=TRUE,text.vert=FALSE)
##forestplot(val,vert=TRUE,add=TRUE)

z <- seq(10)
zu <- c(z[-1],10)
plot(z,type="n")
confband(z,zu,rep(0,length(z)),col=Col("darkblue"),polygon=TRUE,step=TRUE)
confband(z,zu,zu-2,col=Col("darkred"),polygon=TRUE,step=TRUE)

z <- seq(0,1,length.out=100)
plot(z,z,type="n")
confband(z,z,z^2,polygon=TRUE,col="darkred", alpha=0.1)

set.seed(1)
k <- 10
x <- seq(k)
est <- rnorm(k)
sd <- runif(k)
val <- cbind(x,est,est-sd,est+sd)
par(mfrow=c(1,2))
plot(0,type="n",xlim=c(0,k+1),ylim=range(val[,-1]),axes=FALSE,xlab="",ylab="")
axis(2)
confband(val[,1],val[,3],val[,4],val[,2],pch=16,cex=2)
plot(0,type="n",ylim=c(0,k+1),xlim=range(val[,-1]),axes=FALSE,xlab="",ylab="")
axis(1)
confband(val[,1],val[,3],val[,4],val[,2],pch=16,cex=2,vert=FALSE)

x <- seq(0, 3, length.out=20)
y <- cos(x)
yl <- y - 1
yu <- y + 1
plot_region(x, y, yl, yu)
plot_region(x, y, yl, yu, type='s', col="darkblue", add=TRUE)
```

`confint.lvmfit` *Calculate confidence limits for parameters*

Description

Calculate Wald og Likelihood based (profile likelihood) confidence intervals

Usage

```
## S3 method for class 'lvmfit'
confint(
  object,
  parm = seq_len(length(coef(object))),
  level = 0.95,
  profile = FALSE,
  curve = FALSE,
  n = 20,
  interval = NULL,
  lower = TRUE,
  upper = TRUE,
  ...
)
```

Arguments

`object`	`lvm`-object.
`parm`	Index of which parameters to calculate confidence limits for.
`level`	Confidence level
`profile`	Logical expression defining whether to calculate confidence limits via the profile log likelihood
`curve`	if FALSE and profile is TRUE, confidence limits are returned. Otherwise, the profile curve is returned.
`n`	Number of points to evaluate profile log-likelihood in over the interval defined by `interval`
`interval`	Interval over which the profiling is done
`lower`	If FALSE the lower limit will not be estimated (profile intervals only)
`upper`	If FALSE the upper limit will not be estimated (profile intervals only)
`...`	Additional arguments to be passed to the low level functions

Details

Calculates either Wald confidence limits:

$$\hat{\theta} \pm z_{\alpha/2} * \hat{\sigma}_{\hat{\theta}}$$

or profile likelihood confidence limits, defined as the set of value τ:

$$logLik(\hat{\theta}_\tau, \tau) - logLik(\hat{\theta}) < q_\alpha/2$$

where q_α is the α fractile of the χ^2_1 distribution, and $\hat{\theta}_\tau$ are obtained by maximizing the log-likelihood with tau being fixed.

Value

A 2xp matrix with columns of lower and upper confidence limits

Author(s)

Klaus K. Holst

See Also

bootstrap{lvm}

Examples

```
m <- lvm(y~x)
d <- sim(m,100)
e <- estimate(lvm(y~x), d)
confint(e,3,profile=TRUE)
confint(e,3)
 ## Reduce Ex.timings
B <- bootstrap(e,R=50)
B
```

---

confpred *Conformal prediction*

Description

Conformal predicions using locally weighted conformal inference with a split-conformal algorithm

Usage

```
confpred(object, data, newdata = data, alpha = 0.05, mad, ...)
```

Arguments

object	Model object (lm, glm or similar with predict method) or formula (lm)
data	data.frame
newdata	New data.frame to make predictions for
alpha	Level of prediction interval
mad	Conditional model (formula) for the MAD (locally-weighted CP)
...	Additional arguments to lower level functions

Value

data.frame with fitted (fit), lower (lwr) and upper (upr) predictions bands.

Examples

```
set.seed(123)
n <- 200
x <- seq(0,6,length.out=n)
delta <- 3
ss <- exp(-1+1.5*cos((x-delta)))
ee <- rnorm(n,sd=ss)
y <- (x-delta)+3*cos(x+4.5-delta)+ee
d <- data.frame(y=y,x=x)

newd <- data.frame(x=seq(0,6,length.out=50))
cc <- confpred(lm(y~splines::ns(x,knots=c(1,3,5)),data=d), data=d, newdata=newd)
if (interactive()) {
plot(y~x,pch=16,col=lava::Col("black"),ylim=c(-10,10),xlab="X",ylab="Y")
with(cc,
     lava::confband(newd$x,lwr,upr,fit,
        lwd=3,polygon=TRUE,col=Col("blue"),border=FALSE))
}
```

---

constrain<- *Add non-linear constraints to latent variable model*

Description

Add non-linear constraints to latent variable model

Usage

```
## Default S3 replacement method:
constrain(x,par,args,endogenous=TRUE,...) <- value

## S3 replacement method for class 'multigroup'
constrain(x,par,k=1,...) <- value

constraints(object,data=model.frame(object),vcov=object$vcov,level=0.95,
                       p=pars.default(object),k,idx,...)
```

Arguments

x	lvm-object
...	Additional arguments to be passed to the low level functions
value	Real function taking args as a vector argument

par	Name of new parameter. Alternatively a formula with lhs specifying the new parameter and the rhs defining the names of the parameters or variable names defining the new parameter (overruling the args argument).
args	Vector of variables names or parameter names that are used in defining par
endogenous	TRUE if variable is endogenous (sink node)
k	For multigroup models this argument specifies which group to add/extract the constraint
object	lvm-object
data	Data-row from which possible non-linear constraints should be calculated
vcov	Variance matrix of parameter estimates
level	Level of confidence limits
p	Parameter vector
idx	Index indicating which constraints to extract

Details

Add non-linear parameter constraints as well as non-linear associations between covariates and latent or observed variables in the model (non-linear regression).

As an example we will specify the follow multiple regression model:

$$E(Y|X_1, X_2) = \alpha + \beta_1 X_1 + \beta_2 X_2$$
$$V(Y|X_1, X_2) = v$$

which is defined (with the appropiate parameter labels) as

```
m <- lvm(y ~ f(x,beta1) + f(x,beta2))
intercept(m) <- y ~ f(alpha)
covariance(m) <- y ~ f(v)
```

The somewhat strained parameter constraint

$$v = \frac{(beta1 - beta2)^2}{alpha}$$

can then specified as

```
constrain(m,v ~ beta1 + beta2 + alpha) <- function(x) (x[1]-x[2])^2/x[3]
```

A subset of the arguments args can be covariates in the model, allowing the specification of non-linear regression models. As an example the non-linear regression model

$$E(Y \mid X) = \nu + \Phi(\alpha + \beta X)$$

where Φ denotes the standard normal cumulative distribution function, can be defined as

```
m <- lvm(y ~ f(x,0)) # No linear effect of x
```

Next we add three new parameters using the parameter assigment function:

```
parameter(m) <- ~nu+alpha+beta
```

The intercept of Y is defined as mu

`intercept(m) <- y ~ f(mu)`

And finally the newly added intercept parameter mu is defined as the appropiate non-linear function of α, ν and β:

`constrain(m, mu ~ x + alpha + nu) <- function(x) pnorm(x[1]*x[2])+x[3]`

The constraints function can be used to show the estimated non-linear parameter constraints of an estimated model object (lvmfit or multigroupfit). Calling constrain with no additional arguments beyound x will return a list of the functions and parameter names defining the non-linear restrictions.

The gradient function can optionally be added as an attribute grad to the return value of the function defined by value. In this case the analytical derivatives will be calculated via the chain rule when evaluating the corresponding score function of the log-likelihood. If the gradient attribute is omitted the chain rule will be applied on a numeric approximation of the gradient.

Value

A lvm object.

Author(s)

Klaus K. Holst

See Also

regression, intercept, covariance

Examples

```
## Non-linear parameter constraints 1
m <- lvm(y ~ f(x1,gamma)+f(x2,beta))
covariance(m) <- y ~ f(v)
d <- sim(m,100)
m1 <- m; constrain(m1,beta ~ v) <- function(x) x^2
## Define slope of x2 to be the square of the residual variance of y
## Estimate both restricted and unrestricted model
e <- estimate(m,d,control=list(method="NR"))
e1 <- estimate(m1,d)
p1 <- coef(e1)
p1 <- c(p1[1:2],p1[3]^2,p1[3])
## Likelihood of unrestricted model evaluated in MLE of restricted model
logLik(e,p1)
## Likelihood of restricted model (MLE)
logLik(e1)

## Non-linear regression

## Simulate data
m <- lvm(c(y1,y2)~f(x,0)+f(eta,1))
latent(m) <- ~eta
covariance(m,~y1+y2) <- "v"
```

```
intercept(m,~y1+y2) <- "mu"
covariance(m,~eta) <- "zeta"
intercept(m,~eta) <- 0
set.seed(1)
d <- sim(m,100,p=c(v=0.01,zeta=0.01))[,manifest(m)]
d <- transform(d,
               y1=y1+2*pnorm(2*x),
               y2=y2+2*pnorm(2*x))

## Specify model and estimate parameters
constrain(m, mu ~ x + alpha + nu + gamma) <- function(x) x[4]*pnorm(x[3]+x[1]*x[2])
 ## Reduce Ex.Timings
e <- estimate(m,d,control=list(trace=1,constrain=TRUE))
constraints(e,data=d)
## Plot model-fit
plot(y1~x,d,pch=16); points(y2~x,d,pch=16,col="gray")
x0 <- seq(-4,4,length.out=100)
lines(x0,coef(e)["nu"] + coef(e)["gamma"]*pnorm(coef(e)["alpha"]*x0))


## Multigroup model
m1 <- lvm(y ~ f(x,beta)+f(z,beta2))
m2 <- lvm(y ~ f(x,psi) + z)
# simulate data from the two models
d1 <- sim(m1,500)
d2 <- sim(m2,500)
# Add 'non'-linear parameter constraint
constrain(m2,psi ~ beta2) <- function(x) x
## Add parameter beta2 to model 2, now beta2 exists in both models
parameter(m2) <- ~ beta2
ee <- estimate(list(m1,m2),list(d1,d2),control=list(method="NR"))
summary(ee)

m3 <- lvm(y ~ f(x,beta)+f(z,beta2))
m4 <- lvm(y ~ f(x,beta2) + z)
e2 <- estimate(list(m3,m4),list(d1,d2),control=list(method="NR"))
e2
```

`contr` *Create contrast matrix*

Description

Create contrast matrix typically for use with 'estimate' (Wald tests).

Usage

```
contr(p, n, diff = TRUE, ...)
```

Arguments

p	index of non-zero entries (see example)
n	Total number of parameters (if omitted the max number in p will be used)
diff	If FALSE all non-zero entries are +1, otherwise the second non-zero element in each row will be -1.
...	Additional arguments to lower level functions

Examples

```
contr(2,n=5)
contr(as.list(2:4),n=5)
contr(list(1,2,4),n=5)
contr(c(2,3,4),n=5)
contr(list(c(1,3),c(2,4)),n=5)
contr(list(c(1,3),c(2,4),5))

parsedesign(c("aa","b","c"),"?","?",diff=c(FALSE,TRUE))

## All pairs comparisons:
pdiff <- function(n) lava::contr(lapply(seq(n-1), function(x) seq(x, n)))
pdiff(4)
```

---

correlation *Generic method for extracting correlation coefficients of model object*

Description

Generic correlation method

Usage

```
correlation(x, ...)
```

Arguments

x	Object
...	Additional arguments

Author(s)

Klaus K. Holst

`covariance` *Add covariance structure to Latent Variable Model*

Description

Define covariances between residual terms in a `lvm`-object.

Usage

```
## S3 replacement method for class 'lvm'
covariance(object, var1=NULL, var2=NULL, constrain=FALSE, pairwise=FALSE,...) <- value
```

Arguments

`object`	`lvm`-object
`...`	Additional arguments to be passed to the low level functions
`var1`	Vector of variables names (or formula)
`var2`	Vector of variables names (or formula) defining pairwise covariance between `var1` and `var2`)
`constrain`	Define non-linear parameter constraints to ensure positive definite structure
`pairwise`	If TRUE and `var2` is omitted then pairwise correlation is added between all variables in `var1`
`value`	List of parameter values or (if `var1` is unspecified)

Details

The `covariance` function is used to specify correlation structure between residual terms of a latent variable model, using a formula syntax.

For instance, a multivariate model with three response variables,

$$Y_1 = \mu_1 + \epsilon_1$$

$$Y_2 = \mu_2 + \epsilon_2$$

$$Y_3 = \mu_3 + \epsilon_3$$

can be specified as

```
m <- lvm(~y1+y2+y3)
```

Pr. default the two variables are assumed to be independent. To add a covariance parameter $r = cov(\epsilon_1, \epsilon_2)$, we execute the following code

```
covariance(m) <- y1 ~ f(y2,r)
```

The special function `f` and its second argument could be omitted thus assigning an unique parameter the covariance between `y1` and `y2`.

Similarily the marginal variance of the two response variables can be fixed to be identical ($var(Y_i) = v$) via

```
covariance(m) <- c(y1,y2,y3) ~ f(v)
```

To specify a completely unstructured covariance structure, we can call

```
covariance(m) <- ~y1+y2+y3
```

All the parameter values of the linear constraints can be given as the right handside expression of the assigment function `covariance<-` if the first (and possibly second) argument is defined as well. E.g:

```
covariance(m,y1~y1+y2) <- list("a1","b1")
```

```
covariance(m,~y2+y3) <- list("a2",2)
```

Defines

$$var(\epsilon_1) = a1$$

$$var(\epsilon_2) = a2$$

$$var(\epsilon_3) = 2$$

$$cov(\epsilon_1, \epsilon_2) = b1$$

Parameter constraints can be cleared by fixing the relevant parameters to `NA` (see also the `regression` method).

The function `covariance` (called without additional arguments) can be used to inspect the covariance constraints of a `lvm`-object.

Value

A `lvm`-object

Author(s)

Klaus K. Holst

See Also

`regression<-`, `intercept<-`, `constrain<-` `parameter<-`, `latent<-`, `cancel<-`, `kill<-`

Examples

```
m <- lvm()
## Define covariance between residuals terms of y1 and y2
covariance(m) <- y1~y2
covariance(m) <- c(y1,y2)~f(v) ## Same marginal variance
covariance(m) ## Examine covariance structure
```

---

csplit *Split data into folds*

Description

Split data into folds

Usage

```
csplit(x, p = NULL, replace = FALSE, return.index = FALSE, k = 2, ...)
```

Arguments

x	Data or integer (size)
p	Number of folds, or if a number between 0 and 1 is given two folds of size p and (1-p) will be returned
replace	With or with-out replacement
return.index	If TRUE index of folds are returned otherwise the actual data splits are returned (default)
k	(Optional, only used when p=NULL) number of folds without shuffling
...	additional arguments to lower-level functions

Author(s)

Klaus K. Holst

Examples

```
foldr(5,2,rep=2)
csplit(10,3)
csplit(iris[1:10,]) ## Split in two sets 1:(n/2) and (n/2+1):n
csplit(iris[1:10,],0.5)
```

curly *Adds curly brackets to plot*

Description

Adds curly brackets to plot

Usage

```
curly(
  x,
  y,
  len = 1,
  theta = 0,
  wid,
  shape = 1,
  col = 1,
  lwd = 1,
  lty = 1,
  grid = FALSE,
  npoints = 50,
  text = NULL,
  offset = c(0.05, 0)
)
```

Arguments

x	center of the x axis of the curly brackets (or start end coordinates (x1,x2))
y	center of the y axis of the curly brackets (or start end coordinates (y1,y2))
len	Length of the curly brackets
theta	angle (in radians) of the curly brackets orientation
wid	Width of the curly brackets
shape	shape (curvature)
col	color (passed to lines/grid.lines)
lwd	line width (passed to lines/grid.lines)
lty	line type (passed to lines/grid.lines)
grid	If TRUE use grid graphics (compatability with ggplot2)
npoints	Number of points used in curves
text	Label
offset	Label offset (x,y)

Examples

```
if (interactive()) {
plot(0,0,type="n",axes=FALSE,xlab="",ylab="")
curly(x=c(1,0),y=c(0,1),lwd=2,text="a")
curly(x=c(1,0),y=c(0,1),lwd=2,text="b",theta=pi)
curly(x=-0.5,y=0,shape=1,theta=pi,text="c")
curly(x=0,y=0,shape=1,theta=0,text="d")
curly(x=0.5,y=0,len=0.2,theta=pi/2,col="blue",lty=2)
curly(x=0.5,y=-0.5,len=0.2,theta=-pi/2,col="red",shape=1e3,text="e")
}
```

---

deprdiag *50 patients from Monash Medical Centre, Melbourne*

Description

Diagnosis of depression (DSM-III-R MDD, Dysthymia, Adjustment Disorder with Depressed Mood, Depression NOS), Beck Depression Inventory (BDI) (Beck et al., 1961) General Health Questionnaire (GHQ) (Goldberg & Williams, 1988)

Format

data.frame

Source

Clarke, D. M., Smith, G. C., & Herrman, H. E. (1993). A comparative study of screening instruments for mental disorders in general hospital patients. International Journal Psychiatry in Medicine, 23, pp. 323-337.

McKenzie et al. (1996). Comparing correlated Kappas by resampling: Is one level of agreement significantly different from another? J. Psychiat. Res. 30 (6), pp. 483-492.

---

devcoords *Returns device-coordinates and plot-region*

Description

Returns device-coordinates and plot-region

Usage

```
devcoords()
```

Value

A `list` with elements

- `dev.x1`: device left x-coordinate
- `dev.x2`: device right x-coordinate
- `dev.y1`: device bottom y-coordinate
- `dev.y2`: device top y-coordinate
- `fig.x1`: plot left x-coordinate
- `fig.x2`: plot right x-coordinate
- `fig.y1`: plot bottom y-coordinate
- `fig.y2`: plot top y-coordinate

Author(s)

Klaus K. Holst

`diagtest` *Calculate diagnostic tests for 2x2 table*

Description

Calculate prevalence, sensitivity, specificity, and positive and negative predictive values

Usage

```
diagtest(
  table,
  positive = 2,
  exact = FALSE,
  p0 = NA,
  confint = c("logit", "arcsin", "pseudoscore", "exact"),
  ...
)
```

Arguments

`table`	Table or (matrix/data.frame with two columns)
`positive`	Switch reference
`exact`	If TRUE exact binomial proportions CI/test will be used
`p0`	Optional null hypothesis (test prevalenc, sensitivity, ...)
`confint`	Type of confidence limits
`...`	Additional arguments to lower level functions

Details

Table should be in the format with outcome in columns and test in rows. Data.frame should be with test in the first column and outcome in the second column.

Author(s)

Klaus Holst

Examples

```
M <- as.table(matrix(c(42,12,
                       35,28),ncol=2,byrow=TRUE,
                     dimnames=list(rater=c("no","yes"),gold=c("no","yes"))))
diagtest(M,exact=TRUE)
```

---

dsep.lvm *Check d-separation criterion*

Description

Check for conditional independence (d-separation)

Usage

```
## S3 method for class 'lvm'
dsep(object, x, cond = NULL, return.graph = FALSE, ...)
```

Arguments

object	lvm object
x	Variables for which to check for conditional independence
cond	Conditioning set
return.graph	If TRUE the moralized ancestral graph with the conditioning set removed is returned
...	Additional arguments to lower level functions

Details

The argument x can be given as a formula, e.g. x~y|z+v or ~x+y|z+v With everything on the rhs of the bar defining the variables on which to condition on.

Examples

```
m <- lvm(x5 ~ x4+x3, x4~x3+x1, x3~x2, x2~x1)
if (interactive()) {
plot(m,layoutType='neato')
}
dsep(m,x5~x1|x2+x4)
dsep(m,x5~x1|x3+x4)
dsep(m,~x1+x2+x3|x4)
```

`equivalence` *Identify candidates of equivalent models*

Description

Identifies candidates of equivalent models

Usage

```
equivalence(x, rel, tol = 0.001, k = 1, omitrel = TRUE, ...)
```

Arguments

`x`	`lvmfit`-object
`rel`	Formula or character-vector specifying two variables to omit from the model and subsequently search for possible equivalent models
`tol`	Define two models as empirical equivalent if the absolute difference in score test is less than `tol`
`k`	Number of parameters to test simultaneously. For `equivalence` the number of additional associations to be added instead of `rel`.
`omitrel`	if `k` greater than 1, this boolean defines wether to omit candidates containing `rel` from the output
`...`	Additional arguments to be passed to the lower-level functions

Author(s)

Klaus K. Holst

See Also

`compare`, `modelsearch`

estimate.array *Estimate parameters and influence function.*

Description

Estimate parameters for the sample mean, variance, and quantiles

Usage

```
## S3 method for class 'array'
estimate(x, type = "mean", probs = 0.5, ...)
```

Arguments

x	numeric matrix
type	target parameter ("mean", "variance", "quantile")
probs	numeric vector of probabilities (for type="quantile")
...	Additional arguments to lower level functions (i.e., stats::density.default when type="quantile")

Value

Object of class `estimate` (see estimate.default).

estimate.default *Influence function based inference*

Description

Primary tool for obtaining parameter estimates with robust (sandwich) standard errors, applying the delta method, and testing linear hypotheses. The function returns an object of class `estimate` which serves as a general container for parameter estimates and their influence functions (IFs). Three calling conventions are supported:

Usage

```
## Default S3 method:
estimate(
  x = NULL,
  f = NULL,
  ...,
  data,
  id,
  coef,
  IC = TRUE,
```

```
  vcov,
  stack = TRUE,
  average = FALSE,
  subset,
  keep,
  use,
  regex = FALSE,
  ignore.case = FALSE,
  print = NULL,
  labels,
  label.width,
  contrast,
  null,
  level = NULL,
  type = NULL,
  var.adj = NULL,
  df = NULL,
  back.transform = NULL
)
```

Arguments

x	model object (glm, lvmfit, ...) or an existing estimate object. When two model objects are supplied (e.g., estimate(g, g0)) a likelihood-ratio test is performed.
f	transformation of model parameters. Accepts several input types: - A **function** f(p) or f(p, data): applies the delta method. When f returns a named list the names are used as parameter labels. - A **matrix**: used as a contrast (linear combination) matrix. - A **numeric vector** of parameter indices: converted to a contrast that selects and differences those parameters. - A **list** of indices: each element selects one parameter. - **Character** expressions: supports wildcards ("?", "*") and arithmetic on parameter names (e.g., "z" - "x", 2 * "z" - 3 * "x").
...	additional arguments to lower level functions
data	data.frame used by f when the transformation depends on covariates (see average). Defaults to model.frame(x).
id	(optional) cluster identifier. Can be a vector of cluster IDs, a one-sided formula (evaluated in data), a single character column name, or a logical scalar (TRUE for one-to-one matching, FALSE for independence). When supplied, the IF is aggregated within clusters to produce cluster-robust standard errors.
coef	(optional) named parameter vector. Used instead of coef(x) when constructing an estimate object without a model.
IC	if TRUE (default) the influence function matrix is estimated and stored in the returned object (extract with the IC method). Can also be a user-supplied IF matrix (one row per observation, one column per parameter), which is used directly instead of estimating it from x.

vcov — (optional) covariance matrix of parameter estimates, or a logical. If TRUE, stats::vcov is used to obtain the (model-based) covariance matrix from x, yielding non-robust standard errors. If a matrix is supplied it is used directly. When omitted or FALSE, robust standard errors are computed from the influence function.

stack — if TRUE (default) the influence function contributions are summed within each cluster defined by id. Set to FALSE to keep the un-stacked (per-observation) decomposition.

average — if TRUE the function computes the standardized (marginalized) estimate $\hat{\Psi} = P_n f(X; \hat{\theta})$, i.e., the empirical mean of f(p, data) over all rows of data. The influence function accounts for both the empirical averaging and the parameter estimation uncertainty (see Details).

subset — (optional) logical vector, expression evaluated in data, or column name. When used together with average = TRUE, the average is conditioned on the subpopulation where subset is TRUE, yielding a conditional marginalized estimate.

keep — (optional) index of parameters to keep from final result. Accepts integer indices, character names, or (with regex = TRUE) perl-compatible regular expressions.

use — (optional) index of parameters to use in calculations. The selected parameters are first extracted (via keep) and then the remaining arguments (f, contrast, etc.) are applied to this subset.

regex — if TRUE use perl-compatible regular expressions for keep and use arguments

ignore.case — ignore case in regular expressions

print — (optional) custom print function for the resulting estimate object

labels — (optional) character vector of coefficient names

label.width — (optional) max display width of labels

contrast — (deprecated, use summary method) contrast matrix for a final Wald test. When supplied together with null, tests $H_0 : B\theta = b_0$.

null — (deprecated, use summary method) null hypothesis vector b_0 to test against (default 0)

level — (deprecated, use summary method) level of confidence limits (default 0.95)

type — (deprecated, use summary method) type of small-sample correction for cluster-robust variance. One of: - "robust" (default): no correction.

- "df": applies $n/(n-p)$ correction (Mancl & DeRouen, 2001). - "mbn": Morel-Bokossa-Neerchal (2003) correction. - "hc3": leverage-adjusted HC3-type correction (blended with var.adj). - "hc4": Cribari-Neto (2004) leverage-adjusted correction.

var.adj — (deprecated, use summary method) blending parameter for the HC3 leverage adjustment (default 0.25). Controls the weight between observation-level empirical leverage and the average leverage p/n.

df — (deprecated, use summary method) degrees of freedom for t-based inference (default: NULL for Gaussian approximation; when set, confidence intervals and p-values use the t-distribution with df degrees of freedom).

back.transform — (deprecated, use summary method) function applied to the point estimates and confidence interval bounds *after* inference is performed on the original scale. Useful for variance-stabilizing transformations, e.g., compute CIs on the atanh (Fisher z) scale and back-transform with tanh.

Details

- `estimate(x, ...)` – extract estimates from a model object
- `estimate(coef=, IC=, ...)` – construct from coefficients and IF matrix
- `estimate(coef=, vcov=, ...)` – construct from coefficients and covariance matrix

Value

Object of class `estimate` with the following elements:

`coef`	Named vector of parameter estimates.
`vcov`	Variance-covariance matrix.
`IC`	Influence function matrix (observations x parameters).
`coefmat`	Formatted coefficient table (estimate, std.err, confidence limits, p-value).
`id`	Cluster/id variable used.
`ncluster`	Number of clusters.
`n`	Number of observations.
`compare`	(When `null` or contrasts are specified) Wald test result.

Influence functions and robust standard errors

An estimator $\widehat{\theta}$ is *regular and asymptotically linear* (RAL) when it admits the iid decomposition

$$\sqrt{n}(\widehat{\theta} - \theta) = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \mathrm{IC}(Z_i; P) + o_p(1)$$

where IC is the unique *influence function* satisfying $E\{\mathrm{IC}(Z; P)\} = 0$. By the central limit theorem

$$\sqrt{n}(\widehat{\theta} - \theta) \xrightarrow{d} N(0,\ \mathrm{Var}\{\mathrm{IC}(Z; P)\})$$

and the asymptotic variance is consistently estimated by the empirical variance of the plugin IF estimate, yielding robust (sandwich) standard errors. The estimated IF can be extracted with the IC method.

Parameter transformations (delta method)

When `f` is a function $\phi : R^p \to R^m$, the delta method is applied:

$$\sqrt{n}\{\phi(\widehat{\theta}) - \phi(\theta)\} = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \nabla\phi(\theta)\, \mathrm{IC}(Z_i; P) + o_p(1)$$

Derivatives are computed numerically via numDeriv::jacobian unless the function returns an attribute `"grad"` with the analytic Jacobian.

Alternatively, `estimate` objects support direct arithmetic operations (e.g., `a * b`, `exp(a)`, `a^b`) which apply the delta method with *exact* (analytical) derivatives computed automatically. This influence function calculus allows building complex transformations from simple building blocks without numerical differentiation. See the last example section ("influence function calculus") and `vignette("influencefunction", package = "lava")` for details.

Averaging and marginalization

When `average = TRUE` and `f(p, data)` depends on covariates, the target parameter is the standardized (marginalized) estimate $\Psi = E\{f(X;\theta)\}$. The IF for the averaged estimate accounts for both the empirical averaging and parameter estimation uncertainty:

$$\mathrm{IC}_\Psi(Z;P) = f(X;\theta) - \Psi + [E\nabla_\theta f(X;\theta)]\,\phi(Z;P)$$

When `subset` is also specified, the average is conditioned on the subpopulation, yielding a conditional marginalized estimate.

Cluster-robust standard errors

When `id` is supplied, the per-observation IF contributions are summed within clusters (when `stack = TRUE`), producing the cluster-level IF $\widetilde{\mathrm{IC}}(Z_i;P) = \sum_{k=1}^{N_i} \frac{n}{N}\mathrm{IC}(Z_{ik};P)$. The resulting variance estimate is equivalent to the GEE working independence sandwich estimator.

For full theoretical background and worked examples see `vignette("influencefunction", package = "lava")`.

See Also

estimate.array, merge.estimate, contr, parsedesign, pairwise.diff, c.estimate, summary.estimate, `coef.estimate`, `vcov.estimate`, `transform.estimate`, `labels.estimate`,

Examples

```
## Simulation from logistic regression model
m <- lvm(y~x+z);
distribution(m,y~x) <- binomial.lvm("logit")
d <- sim(m,1000)
g <- glm(y~z+x,data=d,family=binomial())
g0 <- glm(y~1,data=d,family=binomial())

## LRT
estimate(g, g0)


estimate(g)

## Testing contrasts
summary(estimate(g), null=0)
estimate(g, rbind(c(1,1,0), c(1,0,2)))
summary(estimate(g, rbind(c(1,1,0), c(1,0,2))), null=c(1,2))
estimate(g, 2:3) ## same as cbind(0,1,-1)
estimate(g, as.list(2:3)) ## same as rbind(c(0,1,0),c(0,0,1))
## Alternative syntax
estimate(g, "z", "z"-"x", 2*"z"-3*"x")
estimate(g, "?")  ## Wildcards
estimate(g, "*Int*", "z")
summary(estimate(g, "1", "2"-"3"), null = c(0,1))
estimate(g, 2, 3)
```

```
## Usual (non-robust) confidence intervals
estimate(g, vcov=TRUE)
estimate(g, vcov=vcov(g))

## Transformations
estimate(g, function(p) p[1]+p[2])

## Multiple parameters
e <- estimate(g, function(p) c(p[1]+p[2], p[1]*p[2]))
e
vcov(e)

## Label new parameters
estimate(g, function(p) list("a1"=p[1]+p[2], "b1"=p[1]*p[2]))
#'
## Multiple group
m <- lvm(y~x)
m <- baptize(m)
d2 <- d1 <- sim(m,50,seed=1)
e <- estimate(list(m,m),list(d1,d2))
estimate(e) ## Wrong
ee <- estimate(e, id=rep(seq(nrow(d1)), 2)) ## Clustered
ee
estimate(lm(y~x,d1))

## Marginalize / standardization
f <- function(p,data)
  list(p0=expit(p["(Intercept)"] + p["z"]*data[,"z"]),
       p1=expit(p["(Intercept)"] + p["x"] + p["z"]*data[,"z"]))
e <- estimate(g, f, average=TRUE)
e
estimate(e,diff)
estimate(e,cbind(1,1))

## Clusters and subset (conditional marginal effects)
d$id <- rep(seq(nrow(d)/4),each=4)
estimate(g,function(p,data)
         list(p0=expit(p[1] + p["z"]*data[,"z"])),
         subset=d$z>0, id=d$id, average=TRUE)

## More examples with clusters:
m <- lvm(c(y1,y2,y3)~u+x)
d <- sim(m,10)
l1 <- glm(y1~x,data=d)
l2 <- glm(y2~x,data=d)
l3 <- glm(y3~x,data=d)

## Some random id-numbers
id1 <- c(1,1,4,1,3,1,2,3,4,5)
id2 <- c(1,2,3,4,5,6,7,8,1,1)
id3 <- seq(10)

## Un-stacked and stacked i.i.d. decomposition
```

```
IC(estimate(l1,id=id1,stack=FALSE))
IC(estimate(l1,id=id1))

## Combined i.i.d. decomposition
e1 <- estimate(l1,id=id1)
e2 <- estimate(l2,id=id2)
e3 <- estimate(l3,id=id3)
(a2 <- merge(e1,e2,e3))

## If all models were estimated on the same data we could use the
## syntax:
## Reduce(merge,estimate(list(l1,l2,l3)))

## Same:
IC(a1 <- merge(l1,l2,l3,id=list(id1,id2,id3)))

IC(merge(l1,l2,l3,id=TRUE)) # one-to-one (same clusters)
IC(merge(l1,l2,l3,id=FALSE)) # independence


# ------ influence function calculus -------
a <- estimate(coef = c("a" = 0.5), IC = scale(rnorm(10), scale=FALSE), id = 1:10)
b <- estimate(coef = c("b" = 0.8), IC = scale(rnorm(10), scale=FALSE), id = 1:10)

e <- c(a, b) # merge
merge(a, b)
c(e1=a, b) # naming of par
labels(e, c("p1", "p2")) # renaming parameters
e["a"] # subset
subset(e, "a")

# pipes
# c(a, b) |>
#  transform(function(x) x^2) |>
#  subset("a") |>
#  labels("sq")

# Parameter transformation with automatic calculation of derivatives
a * b
(3 * cos(a) / sqrt(b) + 1) / a
expit(c(a,b))
c(sum=sum(e), sum2=a+b,
  prod=prod(e), prod2=a*b)
e %*% e # inner prod.
c(1, 2) %*% e
c(pow = a^b)
a^c(0.5, 2)
c(b=e["a"] * e["b"] / a, also.b=e["b"])

B <- rbind(c(1,-1), c(1,0), c(0,1))
B %*% e
e == 1 # wald-test, null-hypothesis H0: b=1
e == c(1,2)
```

```
B %*% e == 1
```

`estimate.formula` *Estimate method for formulas*

Description

Fit a GLM model specified by a formula and apply estimate.default for robust inference.

Usage

```
## S3 method for class 'formula'
estimate(
  x,
  data,
  weights,
  family = stats::gaussian,
  ...,
  model = "glm",
  lvm = FALSE
)
```

Arguments

`x`	formula specifying the model
`data`	data.frame
`weights`	optional weights
`family`	GLM family (default `gaussian`)
`...`	Additional arguments to estimate.default
`model`	character string specifying the model type (default `"glm"`)
`lvm`	logical, if TRUE use `estimate.lvm` instead

Value

Object of class `estimate` (see estimate.default).

estimate.glm *Estimate method for GLM objects*

Description

Applies estimate.default to obtain robust inference for a fitted glm object.

Usage

```
## S3 method for class 'glm'
estimate(x, ...)
```

Arguments

x	Fitted glm object
...	Additional arguments to estimate.default

Value

Object of class estimate (see estimate.default).

estimate.list *Estimate method for lists*

Description

Estimate method for lists

Usage

```
## S3 method for class 'list'
estimate(x, ...)
```

Arguments

x	list of model objects or lvm objects
...	Additional arguments to lower level functions

Value

Object of class estimate.list (a list of estimate objects), or a fitted multi-group model if input is a list of lvm objects.

`estimate.lvm` *Estimation of parameters in a Latent Variable Model (lvm)*

Description

Estimate parameters. MLE, IV or user-defined estimator.

Usage

```
## S3 method for class 'lvm'
estimate(
  x,
  data = parent.frame(),
  estimator = NULL,
  control = list(),
  missing = FALSE,
  weights,
  weightsname,
  data2,
  id,
  fix,
  index = !quick,
  graph = FALSE,
  messages = lava.options()$messages,
  quick = FALSE,
  method,
  param,
  cluster,
  p,
  ...
)
```

Arguments

`x`	`lvm`-object
`data`	`data.frame`
`estimator`	String defining the estimator (see details below)
`control`	control/optimization parameters (see details below)
`missing`	Logical variable indiciating how to treat missing data. Setting to FALSE leads to complete case analysis. In the other case likelihood based inference is obtained by integrating out the missing data under assumption the assumption that data is missing at random (MAR).
`weights`	Optional weights to used by the chosen estimator.
`weightsname`	Weights names (variable names of the model) in case `weights` was given as a vector of column names of `data`

data2	Optional additional dataset used by the chosen estimator.
id	Vector (or name of column in data) that identifies correlated groups of observations in the data leading to variance estimates based on a sandwich estimator
fix	Logical variable indicating whether parameter restriction automatically should be imposed (e.g. intercepts of latent variables set to 0 and at least one regression parameter of each measurement model fixed to ensure identifiability.)
index	For internal use only
graph	For internal use only
messages	Control how much information should be printed during estimation (0: none)
quick	If TRUE the parameter estimates are calculated but all additional information such as standard errors are skipped
method	Optimization method
param	set parametrization (see help(lava.options))
cluster	Obsolete. Alias for 'id'.
p	Evaluate model in parameter 'p' (no optimization)
...	Additional arguments to be passed to lower-level functions

Details

A list of parameters controlling the estimation and optimization procedures is parsed via the control argument. By default Maximum Likelihood is used assuming multivariate normal distributed measurement errors. A list with one or more of the following elements is expected:

start: Starting value. The order of the parameters can be shown by calling coef (with mean=TRUE) on the lvm-object or with plot(..., labels=TRUE). Note that this requires a check that it is actual the model being estimated, as estimate might add additional restriction to the model, e.g. through the fix and exo.fix arguments. The lvm-object of a fitted model can be extracted with the Model-function.

starterfun: Starter-function with syntax function(lvm, S, mu). Three builtin functions are available: startvalues, startvalues0, startvalues1, ...

estimator: String defining which estimator to use (Defaults to "gaussian")

meanstructure Logical variable indicating whether to fit model with meanstructure.

method: String pointing to alternative optimizer (e.g. optim to use simulated annealing).

control: Parameters passed to the optimizer (default stats::nlminb).

tol: Tolerance of optimization constraints on lower limit of variance parameters.

Value

A lvmfit-object.

Author(s)

Klaus K. Holst

See Also

estimate.default score, information

Examples

```
dd <- read.table(header=TRUE,
text="x1 x2 x3
 0.0 -0.5 -2.5
-0.5 -2.0  0.0
 1.0  1.5  1.0
 0.0  0.5  0.0
-2.5 -1.5 -1.0")
e <- estimate(lvm(c(x1,x2,x3)~u),dd)

## Simulation example
m <- lvm(list(y~v1+v2+v3+v4,c(v1,v2,v3,v4)~x))
covariance(m) <- v1~v2+v3+v4
dd <- sim(m,10000) ## Simulate 10000 observations from model
e <- estimate(m, dd) ## Estimate parameters
e

## Using just sufficient statistics
n <- nrow(dd)
e0 <- estimate(m,data=list(S=cov(dd)*(n-1)/n,mu=colMeans(dd),n=n))
rm(dd)

## Multiple group analysis
m <- lvm()
regression(m) <- c(y1,y2,y3)~u
regression(m) <- u~x
d1 <- sim(m,100,p=c("u,u"=1,"u~x"=1))
d2 <- sim(m,100,p=c("u,u"=2,"u~x"=-1))

mm <- baptize(m)
regression(mm,u~x) <- NA
covariance(mm,~u) <- NA
intercept(mm,~u) <- NA
ee <- estimate(list(mm,mm),list(d1,d2))

## Missing data
d0 <- makemissing(d1,cols=1:2)
e0 <- estimate(m,d0,missing=TRUE)
e0
```

---

eventTime *Add an observed event time outcome to a latent variable model.*

Description

For example, if the model 'm' includes latent event time variables are called 'T1' and 'T2' and 'C' is the end of follow-up (right censored), then one can specify

Usage

```
eventTime(object, formula, eventName = "status", ...)
```

Arguments

object	Model object
formula	Formula (see details)
eventName	Event names
...	Additional arguments to lower levels functions

Details

`eventTime(object=m,formula=ObsTime~min(T1=a,T2=b,C=0,"ObsEvent"))`

when data are simulated from the model one gets 2 new columns:

- "ObsTime": the smallest of T1, T2 and C
- "ObsEvent": 'a' if T1 is smallest, 'b' if T2 is smallest and '0' if C is smallest

Note that "ObsEvent" and "ObsTime" are names specified by the user.

Author(s)

Thomas A. Gerds, Klaus K. Holst

Examples

```
# Right censored survival data without covariates
m0 <- lvm()
distribution(m0,"eventtime") <- coxWeibull.lvm(scale=1/100,shape=2)
distribution(m0,"censtime") <- coxExponential.lvm(rate=1/10)
m0 <- eventTime(m0,time~min(eventtime=1,censtime=0),"status")
sim(m0,10)

# Alternative specification of the right censored survival outcome
## eventTime(m,"Status") <- ~min(eventtime=1,censtime=0)

# Cox regression:
# lava implements two different parametrizations of the same
# Weibull regression model. The first specifies
# the effects of covariates as proportional hazard ratios
# and works as follows:
m <- lvm()
distribution(m,"eventtime") <- coxWeibull.lvm(scale=1/100,shape=2)
distribution(m,"censtime") <- coxWeibull.lvm(scale=1/100,shape=2)
m <- eventTime(m,time~min(eventtime=1,censtime=0),"status")
distribution(m,"sex") <- binomial.lvm(p=0.4)
distribution(m,"sbp") <- normal.lvm(mean=120,sd=20)
regression(m,from="sex",to="eventtime") <- 0.4
regression(m,from="sbp",to="eventtime") <- -0.01
sim(m,6)
```

```
# The parameters can be recovered using a Cox regression
# routine or a Weibull regression model. E.g.,
## Not run:
    set.seed(18)
    d <- sim(m,1000)
    library(survival)
    coxph(Surv(time,status)~sex+sbp,data=d)

    sr <- survreg(Surv(time,status)~sex+sbp,data=d)
    library(SurvRegCensCov)
    ConvertWeibull(sr)


## End(Not run)

# The second parametrization is an accelerated failure time
# regression model and uses the function weibull.lvm instead
# of coxWeibull.lvm to specify the event time distributions.
# Here is an example:

ma <- lvm()
distribution(ma,"eventtime") <- weibull.lvm(scale=3,shape=1/0.7)
distribution(ma,"censtime") <- weibull.lvm(scale=2,shape=1/0.7)
ma <- eventTime(ma,time~min(eventtime=1,censtime=0),"status")
distribution(ma,"sex") <- binomial.lvm(p=0.4)
distribution(ma,"sbp") <- normal.lvm(mean=120,sd=20)
regression(ma,from="sex",to="eventtime") <- 0.7
regression(ma,from="sbp",to="eventtime") <- -0.008
set.seed(17)
sim(ma,6)
# The regression coefficients of the AFT model
# can be tranformed into log(hazard ratios):
#  coef.coxWeibull = - coef.weibull / shape.weibull
## Not run:
    set.seed(17)
    da <- sim(ma,1000)
    library(survival)
    fa <- coxph(Surv(time,status)~sex+sbp,data=da)
    coef(fa)
    c(0.7,-0.008)/0.7

## End(Not run)


# The following are equivalent parametrizations
# which produce exactly the same random numbers:

model.aft <- lvm()
distribution(model.aft,"eventtime") <- weibull.lvm(intercept=-log(1/100)/2,sigma=1/2)
distribution(model.aft,"censtime") <- weibull.lvm(intercept=-log(1/100)/2,sigma=1/2)
sim(model.aft,6,seed=17)

model.aft <- lvm()
```

```
distribution(model.aft,"eventtime") <- weibull.lvm(scale=100^(1/2), shape=2)
distribution(model.aft,"censtime") <- weibull.lvm(scale=100^(1/2), shape=2)
sim(model.aft,6,seed=17)

model.cox <- lvm()
distribution(model.cox,"eventtime") <- coxWeibull.lvm(scale=1/100,shape=2)
distribution(model.cox,"censtime") <- coxWeibull.lvm(scale=1/100,shape=2)
sim(model.cox,6,seed=17)

# The minimum of multiple latent times one of them still
# being a censoring time, yield
# right censored competing risks data

mc <- lvm()
distribution(mc,~X2) <- binomial.lvm()
regression(mc) <- T1~f(X1,-.5)+f(X2,0.3)
regression(mc) <- T2~f(X2,0.6)
distribution(mc,~T1) <- coxWeibull.lvm(scale=1/100)
distribution(mc,~T2) <- coxWeibull.lvm(scale=1/100)
distribution(mc,~C) <- coxWeibull.lvm(scale=1/100)
mc <- eventTime(mc,time~min(T1=1,T2=2,C=0),"event")
sim(mc,6)
```

Expand *Create a Data Frame from All Combinations of Factors*

Description

Create a Data Frame from All Combinations of Factors

Usage

```
Expand(`_data`, ...)
```

Arguments

_data	Data.frame
...	vectors, factors or a list containing these

Details

Simple wrapper of the 'expand.grid' function. If x is a table then a data frame is returned with one row pr individual observation.

Author(s)

Klaus K. Holst

Examples

```
dd <- Expand(iris, Sepal.Length=2:8, Species=c("virginica","setosa"))
summary(dd)

T <- with(warpbreaks, table(wool, tension))
Expand(T)
```

`fplot` *fplot*

Description

Faster plot via RGL

Usage

```
fplot(
  x,
  y,
  z = NULL,
  xlab,
  ylab,
  ...,
  z.col = topo.colors(64),
  data = parent.frame(),
  add = FALSE,
  aspect = c(1, 1),
  zoom = 0.8
)
```

Arguments

`x`	X variable
`y`	Y variable
`z`	Z variable (optional)
`xlab`	x-axis label
`ylab`	y-axis label
`...`	additional arggument to lower-level plot functions
`z.col`	color (use argument alpha to set transparency)
`data`	data.frame
`add`	if TRUE use current active device
`aspect`	aspect ratio
`zoom`	zoom level

Examples

```
if (interactive()) {
data(iris)
fplot(Sepal.Length ~ Petal.Length+Species, data=iris, size=2, type="s")
}
```

---

getSAS *Read SAS output*

Description

Run SAS code like in the following:

Usage

```
getSAS(infile, entry = "Parameter Estimates", ...)
```

Arguments

infile	file (csv file generated by ODS)
entry	Name of entry to capture
...	additional arguments to lower level functions

Details

ODS CSVALL BODY="myest.csv"; proc nlmixed data=aj qpoints=2 dampstep=0.5; ... run; ODS CSVALL Close;

and read results into R with:

```
getsas("myest.csv","Parameter Estimates")
```

Author(s)

Klaus K. Holst

gof *Extract model summaries and GOF statistics for model object*

Description

Calculates various GOF statistics for model object including global chi-squared test statistic and AIC. Extract model-specific mean and variance structure, residuals and various predicitions.

Usage

```
gof(object, ...)

## S3 method for class 'lvmfit'
gof(object, chisq=FALSE, level=0.90, rmsea.threshold=0.05,all=FALSE,...)

moments(x,...)

## S3 method for class 'lvm'
moments(x, p, debug=FALSE, conditional=FALSE, data=NULL, latent=FALSE, ...)

## S3 method for class 'lvmfit'
logLik(object, p=coef(object),
                      data=model.frame(object),
                      model=object$estimator,
                      weights=Weights(object),
                      data2=object$data$data2,
                          ...)

## S3 method for class 'lvmfit'
score(x, data=model.frame(x), p=pars(x), model=x$estimator,
                    weights=Weights(x), data2=x$data$data2, ...)

## S3 method for class 'lvmfit'
information(x,p=pars(x),n=x$data$n,data=model.frame(x),
                  model=x$estimator,weights=Weights(x), data2=x$data$data2, ...)
```

Arguments

object	Model object
...	Additional arguments to be passed to the low level functions
x	Model object
p	Parameter vector used to calculate statistics
data	Data.frame to use
latent	If TRUE predictions of latent variables are included in output
data2	Optional second data.frame (only for censored observations)

weights	Optional weight matrix
n	Number of observations
conditional	If TRUE the conditional moments given the covariates are calculated. Otherwise the joint moments are calculated
model	String defining estimator, e.g. "gaussian" (see estimate)
debug	Debugging only
chisq	Boolean indicating whether to calculate chi-squared goodness-of-fit (always TRUE for estimator='gaussian')
level	Level of confidence limits for RMSEA
rmsea.threshold	Which probability to calculate, Pr(RMSEA<rmsea.treshold)
all	Calculate all (ad hoc) FIT indices: TLI, CFI, NFI, SRMR, ...

Value

A htest-object.

Author(s)

Klaus K. Holst

See Also

information, score

Examples

```
m <- lvm(list(y~v1+v2+v3+v4,c(v1,v2,v3,v4)~x))
set.seed(1)
dd <- sim(m,1000)
e <- estimate(m, dd)
gof(e,all=TRUE,rmsea.threshold=0.05,level=0.9)


set.seed(1)
m <- lvm(list(c(y1,y2,y3)~u,y1~x)); latent(m) <- ~u
regression(m,c(y2,y3)~u) <- "b"
d <- sim(m,1000)
e <- estimate(m,d)
rsq(e)
##'
rr <- rsq(e,TRUE)
rr
estimate(rr,contrast=rbind(c(1,-1,0),c(1,0,-1),c(0,1,-1)))
```

Graph *Extract graph*

Description

Extract or replace graph object

Usage

```
Graph(x, ...)

Graph(x, ...) <- value
```

Arguments

x	Model object
...	Additional arguments to be passed to the low level functions
value	New graphNEL object

Author(s)

Klaus K. Holst

See Also

Model

Examples

```
m <- lvm(y~x)
Graph(m)
```

Grep *Finds elements in vector or column-names in data.frame/matrix*

Description

Pattern matching in a vector or column names of a data.frame or matrix.

Usage

```
Grep(x, pattern, subset = TRUE, ignore.case = TRUE, ...)
```

Arguments

x	vector, matrix or data.frame.
pattern	regular expression to search for
subset	If TRUE returns subset of data.frame/matrix otherwise just the matching column names
ignore.case	Default ignore case
...	Additional arguments to 'grep'

Value

A data.frame with 2 columns with the indices in the first and the matching names in the second.

Author(s)

Klaus K. Holst

See Also

grep, and agrep for approximate string matching,

Examples

```
data(iris)
head(Grep(iris,"(len)|(sp)"))
```

---

hubble *Hubble data*

Description

Velocity (v) and distance (D) measures of 36 Type Ia super-novae from the Hubble Space Telescope

Format

data.frame

Source

Freedman, W. L., et al. 2001, AstroPhysicalJournal, 553, 47.

hubble2 *Hubble data*

Description

Hubble data

Format

data.frame

See Also

hubble

IC.default *Extract influence function from model object*

Description

Extract i.i.d. decomposition (influence function) from model object

Usage

```
## Default S3 method:
IC(x, bread, id = NULL, ...)
```

Arguments

x	model object
bread	(optional) Inverse of derivative of mean score function
id	(optional) id/cluster variable
...	additional arguments

Value

Matrix with rows corresponding to observations and columns to parameters, representing the estimated influence function. Attributes include bread (derivative of the estimating equation).

See Also

var_ic, iid

Examples

```
m <- lvm(y~x+z)
distribution(m, ~y+z) <- binomial.lvm("logit")
d <- sim(m,1e3)
g <- glm(y~x+z,data=d,family=binomial)
var_ic(IC(g))
```

`iid` *Extract i.i.d. decomposition from model object*

Description

This function extracts

Usage

```
iid(x, ...)
```

Arguments

`x`	Model object
`...`	Additional arguments (see the man-page of the IC method)

Value

Matrix (n x p) with the i.i.d. decomposition of the estimator (scaled influence function, i.e., the influence function divided by n).

`images` *Organize several image calls (for visualizing categorical data)*

Description

Visualize categorical by group variable

Usage

```
images(
  x,
  group,
  ncol = 2,
  byrow = TRUE,
  colorbar = 1,
  colorbar.space = 0.1,
  label.offset = 0.02,
```

```
  order = TRUE,
  colorbar.border = 0,
  main,
  rowcol = FALSE,
  plotfun = NULL,
  axis1,
  axis2,
  mar,
 col = list(c("#EFF3FF", "#BDD7E7", "#6BAED6", "#2171B5"), c("#FEE5D9", "#FCAE91",
   "#FB6A4A", "#CB181D"), c("#EDF8E9", "#BAE4B3", "#74C476", "#238B45"), c("#FEEDDE",
    "#FDBE85", "#FD8D3C", "#D94701")),
  ...
)
```

Arguments

x	data.frame or matrix
group	group variable
ncol	number of columns in layout
byrow	organize by row if TRUE
colorbar	Add color bar
colorbar.space	Space around color bar
label.offset	label offset
order	order
colorbar.border	Add border around color bar
main	Main title
rowcol	switch rows and columns
plotfun	Alternative plot function (instead of 'image')
axis1	Axis 1
axis2	Axis 2
mar	Margins
col	Colours
...	Additional arguments to lower level graphics functions

Author(s)

Klaus Holst

Examples

```
X <- matrix(rbinom(400,3,0.5),20)
group <- rep(1:4,each=5)
images(X,colorbar=0,zlim=c(0,3))
images(X,group=group,zlim=c(0,3))
```

```
## Not run:
images(X,group=group,col=list(RColorBrewer::brewer.pal(4,"Purples"),
                              RColorBrewer::brewer.pal(4,"Greys"),
                              RColorBrewer::brewer.pal(4,"YlGn"),
                            RColorBrewer::brewer.pal(4,"PuBuGn")),colorbar=2,zlim=c(0,3))

## End(Not run)
images(list(X,X,X,X),group=group,zlim=c(0,3))
images(list(X,X,X,X),ncol=1,group=group,zlim=c(0,3))
images(list(X,X),group,axis2=c(FALSE,FALSE),axis1=c(FALSE,FALSE),
      mar=list(c(0,0,0,0),c(0,0,0,0)),yaxs="i",xaxs="i",zlim=c(0,3))
```

`index` *Generic method for extract index of an object*

Description

Generic method for extract index of an object

Generic method for setting the index of an object

Usage

```
index(x, ...)

index(x, ...) <- value
```

Arguments

`x`	object on which to set the index
`...`	further arguments to be passed to or from other methods.
`value`	the new index

See Also

index<-

index

`index.lvm` *Extract the parameter indicies of a lvm object*

Description

Extracts the matrices with indices of model parameters from a latent variable model (`lvm`). Returns a list with

- A: Matrix with fixed parameters and ones where parameters are free
- J: Manifest variable selection matrix
- M0: Index of free regression parameters
- M1: Index of free and *unique* regression parameters
- P: Matrix with fixed variance parameters and ones where parameters are free
- P0: Index of free variance parameters
- P1: Index of free and *unique* regression parameters
- npar.var: Number of covariance parameters

Usage

```
## S3 method for class 'lvm'
index(x, ...)

## S3 replacement method for class 'lvm'
index(x, ...) <- value
```

Arguments

`x`	object on which to set the index
`...`	further arguments to be passed to or from other methods.
`value`	new index

See Also

`modelPar()`

indoorenv *Data*

Description

Description

Format

data.frame

Source

Simulated

information *Extract information matrix*

Description

Extract the (observed or expected) information matrix from a model object.

Usage

```
information(x, ...)
```

Arguments

x	Model object
...	Additional arguments to lower level functions

Value

Information matrix or its inverse (depending on arguments).

`intercept` *Fix mean parameters in 'lvm'-object*

Description

Define linear constraints on intercept parameters in a `lvm`-object.

Usage

```
## S3 replacement method for class 'lvm'
intercept(object, vars, ...) <- value
```

Arguments

`object`	`lvm`-object
`...`	Additional arguments
`vars`	character vector of variable names
`value`	Vector (or list) of parameter values or labels (numeric or character) or a formula defining the linear constraints (see also the `regression` or `covariance` methods).

Details

The `intercept` function is used to specify linear constraints on the intercept parameters of a latent variable model. As an example we look at the multivariate regression model

$$E(Y_1|X) = \alpha_1 + \beta_1 X$$
$$E(Y_2|X) = \alpha_2 + \beta_2 X$$

defined by the call

`m <- lvm(c(y1,y2) ~ x)`

To fix $\alpha_1 = \alpha_2$ we call

`intercept(m) <- c(y1,y2) ~ f(mu)`

Fixed parameters can be reset by fixing them to `NA`. For instance to free the parameter restriction of Y_1 and at the same time fixing $\alpha_2 = 2$, we call

`intercept(m, ~y1+y2) <- list(NA,2)`

Calling `intercept` with no additional arguments will return the current intercept restrictions of the `lvm`-object.

Value

A `lvm`-object

Note

Variables will be added to the model if not already present.

Author(s)

Klaus K. Holst

See Also

covariance<-, regression<-, constrain<-, parameter<-, latent<-, cancel<-, kill<-

Examples

```
## A multivariate model
m <- lvm(c(y1,y2) ~ f(x1,beta)+x2)
regression(m) <- y3 ~ f(x1,beta)
intercept(m) <- y1 ~ f(mu)
intercept(m, ~y2+y3) <- list(2,"mu")
intercept(m) ## Examine intercepts of model (NA translates to free/unique paramete##r)
```

---

intervention.lvm *Define intervention*

Description

Define intervention in a lvm object

Usage

```
## S3 method for class 'lvm'
intervention(object, to, value, dist = none.lvm(), ...)
```

Arguments

object	lvm object
to	String defining variable or formula
value	function defining intervention
dist	Distribution
...	Additional arguments to lower level functions

See Also

regression lvm sim

Examples

```
m <- lvm(y ~ a + x, a ~ x)
distribution(m, ~a+y) <- binomial.lvm()
mm <- intervention(m, "a", value=3)
sim(mm, 10)
mm <- intervention(m, a~x, function(x) (x>0)*1)
sim(mm, 10)
```

`Inverse` *Generalized matrix inverse*

Description

Generalized matrix inverse

Usage

```
Inverse(
  X,
  tol = lava.options()$itol,
  det = TRUE,
  names = !chol,
  chol = FALSE,
  symmetric = FALSE
)
```

Arguments

`X`	nxn matrix
`tol`	tolerance for pseudo inverse
`det`	logical, if true the determinant is returned
`names`	preserve dimnames
`chol`	use Cholesky decomposition for calculating inverse otherwise SVD
`symmetric`	set to true if matrix is symmetric

Value

Matrix (generalized inverse of `X`). Attributes include `det` (determinant), `pseudo` (logical, whether pseudo-inverse was used), and `minSV` (minimum singular value).

ksmooth2 *Plot/estimate surface*

Description

Plot/estimate surface

Usage

```
ksmooth2(
  x,
  data,
  h = NULL,
  xlab = NULL,
  ylab = NULL,
  zlab = "",
  gridsize = rep(51L, 2),
  ...
)
```

Arguments

x	formula or data
data	data.frame
h	bandwidth
xlab	X label
ylab	Y label
zlab	Z label
gridsize	grid size of kernel smoother
...	Additional arguments to graphics routine (persp3d or persp)

Examples

```
if (requireNamespace("KernSmooth")) {##'
ksmooth2(rmvn0(1e4,sigma=diag(2)*.5+.5),c(-3.5,3.5),h=1,
       rgl=FALSE,theta=30)
##'
if (interactive()) {
    ksmooth2(rmvn0(1e4,sigma=diag(2)*.5+.5),c(-3.5,3.5),h=1)
    ksmooth2(function(x,y) x^2+y^2, c(-20,20))
    ksmooth2(function(x,y) x^2+y^2, xlim=c(-5,5), ylim=c(0,10))

    f <- function(x,y) 1-sqrt(x^2+y^2)
    surface(f,xlim=c(-1,1),alpha=0.9,aspect=c(1,1,0.75))
    surface(f,xlim=c(-1,1),clut=heat.colors(128))
    ##play3d(spin3d(axis=c(0,0,1), rpm=8), duration=5)
```

```
}

if (interactive()) {
   surface(function(x) dmvn0(x,sigma=diag(2)),c(-3,3),lit=FALSE,smooth=FALSE,box=FALSE,alpha=0.8)
    surface(function(x) dmvn0(x,sigma=diag(2)),c(-3,3),box=FALSE,specular="black")##'
}

if (!inherits(try(find.package("fields"),silent=TRUE),"try-error")) {
    f <- function(x,y) 1-sqrt(x^2+y^2)
    ksmooth2(f,c(-1,1),rgl=FALSE,image=fields::image.plot)
}

}
```

`labels<-` *Define labels of graph*

Description

Alters labels of nodes and edges in the graph of a latent variable model

Usage

```
## Default S3 replacement method:
labels(object, ...) <- value
## S3 replacement method for class 'lvm'
edgelabels(object, to, ...) <- value
## Default S3 replacement method:
nodecolor(object, var=vars(object),
border, labcol, shape, lwd, ...) <- value
```

Arguments

`object`	`lvm`-object.
`...`	Additional arguments (`lwd`, `cex`, `col`, `labcol`), `border`.
`value`	node label/edge label/color
`to`	Formula specifying outcomes and predictors defining relevant edges.
`var`	Formula or character vector specifying the nodes/variables to alter.
`border`	Colors of borders
`labcol`	Text label colors
`shape`	Shape of node
`lwd`	Line width of border

Author(s)

Klaus K. Holst

Examples

```
m <- lvm(c(y,v)~x+z)
regression(m) <- c(v,x)~z
labels(m) <- c(y=expression(psi), z=expression(zeta))
nodecolor(m,~y+z+x,border=c("white","white","black"),
          labcol="white", lwd=c(1,1,5),
          lty=c(1,2)) <-  c("orange","indianred","lightgreen")
edgelabels(m,y~z+x, cex=c(2,1.5), col=c("orange","black"),labcol="darkblue",
           arrowhead=c("tee","dot"),
           lwd=c(3,1)) <- expression(phi,rho)
edgelabels(m,c(v,x)~z, labcol="red", cex=0.8,arrowhead="none") <- 2
if (interactive()) {
    plot(m,addstyle=FALSE)
}

m <- lvm(y~x)
labels(m) <- list(x="multiple\nlines")
if (interactive()) {
op <- par(mfrow=c(1,2))
plot(m,plain=TRUE)
plot(m)
par(op)

d <- sim(m,100)
e <- estimate(m,d)
plot(e,type="sd")
}
```

`lava.options` *Set global options for* `lava`

Description

Extract and set global parameters of `lava`. In particular optimization parameters for the `estimate` function.

Usage

```
lava.options(...)
```

Arguments

`...`	Arguments

Details

- `param`: 'relative' (factor loading and variance of one endogenous variables in each measurement model are fixed to one), 'absolute' (mean and variance of latent variables are set to 0 and 1, respectively), 'hybrid' (intercept of latent variables is fixed to 0, and factor loading

of at least one endogenous variable in each measurement model is fixed to 1), 'none' (no constraints are added)

- `layout`: One of 'dot','fdp','circo','twopi','neato','osage'
- `messages`: Set to 0 to disable various output messages
- ...

see `control` parameter of the `estimate` function.

Value

`list` of parameters

Author(s)

Klaus K. Holst

Examples

```
## Not run:
lava.options(iter.max=100,messages=0)

## End(Not run)
```

---

lvm *Initialize new latent variable model*

Description

Function that constructs a new latent variable model object

Usage

```
lvm(x = NULL, ..., latent = NULL, messages = lava.options()$messages)
```

Arguments

`x`	Vector of variable names. Optional but gives control of the sequence of appearance of the variables. The argument can be given as a character vector or formula, e.g. `~y1+y2` is equivalent to `c("y1","y2")`. Alternatively the argument can be a formula specifying a linear model.
`...`	Additional arguments to be passed to the low level functions
`latent`	(optional) Latent variables
`messages`	Controls what messages are printed (0: none)

Value

Returns an object of class `lvm`.

Author(s)

Klaus K. Holst

See Also

regression, covariance, intercept, ...

Examples

```
m <- lvm() # Empty model
m1 <- lvm(y~x) # Simple linear regression
m2 <- lvm(~y1+y2) # Model with two independent variables (argument)
m3 <- lvm(list(c(y1,y2,y3)~u,u~x+z)) # SEM with three items
```

---

makemissing *Create random missing data*

Description

Generates missing entries in data.frame/matrix

Usage

```
makemissing(
  data,
  p = 0.2,
  cols = seq_len(ncol(data)),
  rowwise = FALSE,
  nafun = function(x) x,
  seed = NULL
)
```

Arguments

data	data.frame
p	Fraction of missing data in each column
cols	Which columns (name or index) to alter
rowwise	Should missing occur row-wise (either none or all selected columns are missing)
nafun	(Optional) function to be applied on data.frame before return (e.g. na.omit to return complete-cases only)
seed	Random seed

Value

data.frame

Author(s)

Klaus K. Holst

`measurement.error` *Two-stage (non-linear) measurement error*

Description

Two-stage measurement error

Usage

```
measurement.error(
  model1,
  formula,
  data = parent.frame(),
  predictfun = function(mu, var, data, ...) mu[, 1]^2 + var[1],
  id1,
  id2,
  ...
)
```

Arguments

`model1`	Stage 1 model
`formula`	Formula specifying observed covariates in stage 2 model
`data`	data.frame
`predictfun`	Predictions to be used in stage 2
`id1`	Optional id-vector of stage 1
`id2`	Optional id-vector of stage 2
`...`	Additional arguments to lower level functions

See Also

stack.estimate

Examples

```
m <- lvm(c(y1,y2,y3)~u,c(y3,y4,y5)~v,u~~v,c(u,v)~x)
transform(m,u2~u) <- function(x) x^2
transform(m,uv~u+v) <- prod
regression(m) <- z~u2+u+v+uv+x
set.seed(1)
d <- sim(m,1000,p=c("u,u"=1))

## Stage 1
```

```
m1 <- lvm(c(y1[0:s],y2[0:s],y3[0:s])~1*u,c(y3[0:s],y4[0:s],y5[0:s])~1*v,u~b*x,u~~v)
latent(m1) <- ~u+v
e1 <- estimate(m1,d)

pp <- function(mu,var,data,...) {
    cbind(u=mu[,"u"],u2=mu[,"u"]^2+var["u","u"],
          v=mu[,"v"],uv=mu[,"u"]*mu[,"v"]+var["u","v"])
}
(e <- measurement.error(e1, z~1+x, data=d, predictfun=pp))

## uu <- seq(-1,1,length.out=100)
## pp <- estimate(e,function(p,...) p["(Intercept)"]+p["u"]*uu+p["u2"]*uu^2)$coefmat
if (interactive()) {
    plot(e,intercept=TRUE,line=0)
    dev.off()
    f <- function(p) p[1]+p["u"]*u+p["u2"]*u^2
    u <- seq(-1,1,length.out=100)
    plot(e, f, data=data.frame(u), ylim=c(-.5,2.5))
}
```

`merge.estimate` *Merge estimate objects*

Description

Merge estimate objects

Usage

```
## S3 method for class 'estimate'
merge(
  x,
  y,
  ...,
  id,
  paired = FALSE,
  labels = NULL,
  keep = NULL,
  subset = NULL,
  regex = FALSE,
  sep = FALSE,
  drop.ic = FALSE,
  ignore.case = FALSE
)
```

Arguments

`x`	Object of class `estimate`

y	Object of class estimate
...	Additional estimate objects or arguments
id	Optional cluster variable
paired	If TRUE a paired (matched) analysis is performed
labels	Optional character vector of labels for the merged estimates
keep	Optional character vector of parameter names to keep
subset	Optional character vector of parameter names to subset
regex	If TRUE, keep and subset are treated as regular expressions
sep	Separator used for labeling
drop.ic	If TRUE, drop the influence function from the result
ignore.case	If TRUE, case is ignored in keep/subset matching

Value

Object of class estimate (see estimate.default).

See Also

c.estimate()

Missing *Missing value generator*

Description

Missing value generator

Usage

```
Missing(object, formula, Rformula, missing.name, suffix = "0", ...)
```

Arguments

object	lvm-object.
formula	The right hand side specifies the name of a latent variable which is not always observed. The left hand side specifies the name of a new variable which is equal to the latent variable but has missing values. If given as a string then this is used as the name of the latent (full-data) name, and the observed data name is 'missing.data'
Rformula	Missing data mechanism with left hand side specifying the name of the observed data indicator (may also just be given as a character instead of a formula)
missing.name	Name of observed data variable (only used if 'formula' was given as a character specifying the name of the full-data variable)
suffix	If missing.name is missing, then the name of the oberved data variable will be the name of the full-data variable + the suffix
...	Passed to binomial.lvm.

Details

This function adds a binary variable to a given `lvm` model and also a variable which is equal to the original variable where the binary variable is equal to zero

Value

lvm object

Author(s)

Thomas A. Gerds tag@biostat.ku.dk

Examples

```
library(lava)
set.seed(17)
m <- lvm(y0~x01+x02+x03)
m <- Missing(m,formula=x1~x01,Rformula=R1~0.3*x02+-0.7*x01,p=0.4)
sim(m,10)


m <- lvm(y~1)
m <- Missing(m,"y","r")
## same as
## m <- Missing(m,y~1,r~1)
sim(m,10)

## same as
m <- lvm(y~1)
Missing(m,"y") <- r~x
sim(m,10)

m <- lvm(y~1)
m <- Missing(m,"y","r",suffix=".")
## same as
## m <- Missing(m,"y","r",missing.name="y.")
## same as
## m <- Missing(m,y.~y,"r")
sim(m,10)
```

`missingdata` *Missing data example*

Description

Simulated data generated from model

$$E(Y_i \mid X) = X, \quad cov(Y_1, Y_2 \mid X) = 0.5$$

Format

list of data.frames

Details

The list contains four data sets

1. Complete data
2. MCAR
3. MAR
4. MNAR (missing mechanism depends on variable V correlated with Y1,Y2)

Source

Simulated

Examples

```
data(missingdata)
e0 <- estimate(lvm(c(y1,y2)~b*x,y1~~y2),missingdata[[1]]) ## No missing
e1 <- estimate(lvm(c(y1,y2)~b*x,y1~~y2),missingdata[[2]]) ## CC (MCAR)
e2 <- estimate(lvm(c(y1,y2)~b*x,y1~~y2),missingdata[[2]],missing=TRUE) ## MCAR
e3 <- estimate(lvm(c(y1,y2)~b*x,y1~~y2),missingdata[[3]]) ## CC (MAR)
e4 <- estimate(lvm(c(y1,y2)~b*x,y1~~y2),missingdata[[3]],missing=TRUE) ## MAR
```

---

mixture *Estimate mixture latent variable model.*

Description

Estimate mixture latent variable model

Usage

```
mixture(
  x,
  data,
  k = length(x),
  control = list(),
  vcov = "observed",
  names = FALSE,
  ...
)
```

Arguments

x	List of lvm objects. If only a single lvm object is given, then a k-mixture of this model is fitted (free parameters varying between mixture components).
data	data.frame
k	Number of mixture components
control	Optimization parameters (see details) #type Type of EM algorithm (standard, classification, stochastic)
vcov	of asymptotic covariance matrix (NULL to omit)
names	If TRUE returns the names of the parameters (for defining starting values)
...	Additional arguments parsed to lower-level functions

Details

Estimate parameters in a mixture of latent variable models via the EM algorithm.

The performance of the EM algorithm can be tuned via the control argument, a list where a subset of the following members can be altered:

start Optional starting values

nstart Evaluate nstart different starting values and run the EM-algorithm on the parameters with largest likelihood

tol Convergence tolerance of the EM-algorithm. The algorithm is stopped when the absolute change in likelihood and parameter (2-norm) between successive iterations is less than tol

iter.max Maximum number of iterations of the EM-algorithm

gamma Scale-down (i.e. number between 0 and 1) of the step-size of the Newton-Raphson algorithm in the M-step

trace Trace information on the EM-algorithm is printed on every traceth iteration

Note that the algorithm can be aborted any time (C-c) and still be saved (via on.exit call).

Author(s)

Klaus K. Holst

See Also

mvnmix

Examples

```
m0 <- lvm(list(y~x+z,x~z))
distribution(m0,~z) <- binomial.lvm()
d <- sim(m0,2000,p=c("y~z"=2,"y~x"=1),seed=1)

## unmeasured confounder example
m <- baptize(lvm(y~x, x~1));
```

```
intercept(m,~x+y) <- NA

if (requireNamespace('mets', quietly=TRUE)) {
  set.seed(42)
  M <- mixture(m,k=2,data=d,control=list(trace=1,tol=1e-6))
  summary(M)
  lm(y~x,d)
  estimate(M,"y~x")
  ## True slope := 1
}
```

`Model` *Extract model*

Description

Extract or replace model object

Usage

```
Model(x, ...)

Model(x, ...) <- value
```

Arguments

`x`	Fitted model
`...`	Additional arguments to be passed to the low level functions
`value`	New model object (e.g. `lvm` or `multigroup`)

Value

Returns a model object (e.g. `lvm` or `multigroup`)

Author(s)

Klaus K. Holst

See Also

`Graph`

Examples

```
m <- lvm(y~x)
e <- estimate(m, sim(m,100))
Model(e)
```

`modelsearch` *Model searching*

Description

Performs Wald or score tests

Usage

```
modelsearch(x, k = 1, dir = "forward", type = "all", ...)
```

Arguments

`x`	`lvmfit`-object
`k`	Number of parameters to test simultaneously. For `equivalence` the number of additional associations to be added instead of `rel`.
`dir`	Direction to do model search. "forward" := add associations/arrows to model/graph (score tests), "backward" := remove associations/arrows from model/graph (wald test)
`type`	If equal to 'correlation' only consider score tests for covariance parameters. If equal to 'regression' go through direct effects only (default 'all' is to do both)
`...`	Additional arguments to be passed to the low level functions

Value

Matrix of test-statistics and p-values

Author(s)

Klaus K. Holst

See Also

`compare`, `equivalence`

Examples

```
m <- lvm();
regression(m) <- c(y1,y2,y3) ~ eta; latent(m) <- ~eta
regression(m) <- eta ~ x
m0 <- m; regression(m0) <- y2 ~ x
dd <- sim(m0,100)[,manifest(m0)]
e <- estimate(m,dd);
modelsearch(e,messages=0)
modelsearch(e,messages=0,type="cor")
```

`multinomial` *Estimate probabilities in contingency table*

Description

Estimate probabilities in contingency table

Usage

```
multinomial(
  x,
  data = parent.frame(),
  marginal = FALSE,
  transform,
  vcov = TRUE,
  IC = TRUE,
  ...
)
```

Arguments

`x`	Formula (or matrix or data.frame with observations, 1 or 2 columns)
`data`	Optional data.frame
`marginal`	If TRUE the marginals are estimated
`transform`	Optional transformation of parameters (e.g., logit)
`vcov`	Calculate asymptotic variance (default TRUE)
`IC`	Return ic decomposition (default TRUE)
`...`	Additional arguments to lower-level functions

Author(s)

Klaus K. Holst

Examples

```
set.seed(1)
breaks <- c(-Inf,-1,0,Inf)
m <- lvm(); covariance(m,pairwise=TRUE) <- ~y1+y2+y3+y4
d <- transform(sim(m,5e2),
               z1=cut(y1,breaks=breaks),
               z2=cut(y2,breaks=breaks),
               z3=cut(y3,breaks=breaks),
               z4=cut(y4,breaks=breaks))

multinomial(d[,5])
(a1 <- multinomial(d[,5:6]))
```

```
(K1 <- kappa(a1)) ## Cohen's kappa

K2 <- kappa(d[,7:8])
## Testing difference K1-K2:
estimate(merge(K1,K2,id=TRUE),diff)

estimate(merge(K1,K2,id=FALSE),diff) ## Wrong std.err ignoring dependence
sqrt(vcov(K1)+vcov(K2))

## Average of the two kappas:
estimate(merge(K1,K2,id=TRUE),function(x) mean(x))
estimate(merge(K1,K2,id=FALSE),function(x) mean(x)) ## Independence
##'
## Goodman-Kruskal's gamma
m2 <- lvm(); covariance(m2) <- y1~y2
breaks1 <- c(-Inf,-1,0,Inf)
breaks2 <- c(-Inf,0,Inf)
d2 <- transform(sim(m2,5e2),
              z1=cut(y1,breaks=breaks1),
              z2=cut(y2,breaks=breaks2))

(g1 <- gkgamma(d2[,3:4]))
## same as
## Not run:
gkgamma(table(d2[,3:4]))
gkgamma(multinomial(d2[,3:4]))

## End(Not run)

##partial gamma
d2$x <- rbinom(nrow(d2),2,0.5)
gkgamma(z1~z2|x,data=d2)
```

`mvnmix` *Estimate mixture latent variable model*

Description

Estimate mixture latent variable model

Usage

```
mvnmix(
  data,
  k = 2,
  theta,
  steps = 500,
  tol = 1e-16,
  lambda = 0,
  mu = NULL,
```

```
  silent = TRUE,
  extra = FALSE,
  n.start = 1,
  init = "kmpp",
  ...
)
```

Arguments

data	data.frame
k	Number of mixture components
theta	Optional starting values
steps	Maximum number of iterations
tol	Convergence tolerance of EM algorithm
lambda	Regularisation parameter. Added to diagonal of covariance matrix (to avoid singularities)
mu	Initial centres (if unspecified random centres will be chosen)
silent	Turn on/off output messages
extra	Extra debug information
n.start	Number of restarts
init	Function to choose initial centres
...	Additional arguments parsed to lower-level functions

Details

Estimate parameters in a mixture of latent variable models via the EM algorithm.

Value

A mixture object

Author(s)

Klaus K. Holst

See Also

mixture

Examples

```
data(faithful)
set.seed(1)
M1 <- mvnmix(faithful[,"waiting",drop=FALSE],k=2)
M2 <- mvnmix(faithful,k=2)
if (interactive()) {
    par(mfrow=c(2,1))
```

```
    plot(M1,col=c("orange","blue"),ylim=c(0,0.05))
    plot(M2,col=c("orange","blue"))
}
```

---

na.pass0 *Handle Missing Values in Objects*

Description

Returns the object with missing data replaced by zeros. This is sometimes useful for example when working with inverse probability weighting of the complete-case data.

Usage

```
na.pass0(object, na.action = na.omit, ...)
```

Arguments

object	data.frame, vector
na.action	function used to identify missing values
...	additional arguments to lower level functions

See Also

na.pass(), na.omit(), na.fail()

Examples

```
d <- data.frame(y=c(1,1,NA,2,NA,2), r=c(1,1,0,1,1,1))
na.pass0(d)
glm(y ~ 1, weights=d$r, data=d, na.action=na.pass0)
```

NA2x *Convert to/from NA*

Description

Convert vector to/from NA

Usage

```
NA2x(s, x = 0)
```

Arguments

s	The input vector (of arbitrary class)
x	The elements to transform into NA resp. what to transform NA into.

Value

A vector with same dimension and class as s.

Author(s)

Klaus K. Holst

Examples

```
##'
x2NA(1:10, 1:5)
NA2x(x2NA(c(1:10),5),5)##'
```

---

nldata *Example data (nonlinear model)*

Description

Example data (nonlinear model)

Format

data.frame

Source

Simulated

NR *Newton-Raphson method*

Description

Newton-Raphson method

Usage

```
NR(
  start,
  objective = NULL,
  gradient = NULL,
  hessian = NULL,
  control,
  args = NULL,
  ...
)
```

Arguments

start	Starting value
objective	Optional objective function (used for selecting step length)
gradient	gradient
hessian	hessian (if NULL a numerical derivative is used)
control	optimization arguments (see details)
args	Optional list of arguments parsed to objective, gradient and hessian
...	additional arguments parsed to lower level functions

Details

control should be a list with one or more of the following components:

- trace integer for which output is printed each 'trace'th iteration
- iter.max number of iterations
- stepsize: Step size (default 1)
- nstepsize: Increase stepsize every nstepsize iteration (from stepsize to 1)
- tol: Convergence criterion (gradient)
- epsilon: threshold used in pseudo-inverse
- backtrack: In each iteration reduce stepsize unless solution is improved according to criterion (gradient, armijo, curvature, wolfe)

Examples

```
# Objective function with gradient and hessian as attributes
f <- function(z) {
   x <- z[1]; y <- z[2]
   val <- x^2 + x*y^2 + x + y
   structure(val, gradient=c(2*x+y^2+1, 2*y*x+1),
             hessian=rbind(c(2,2*y),c(2*y,2*x)))
}
NR(c(0,0),f)

# Parsing arguments to the function and
```

```
g <- function(x,y) (x*y+1)^2
NR(0, gradient=g, args=list(y=2), control=list(trace=1,tol=1e-20))
```

---

nsem *Example SEM data (nonlinear)*

Description

Simulated data

Format

data.frame

Source

Simulated

---

ordinal<- *Define variables as ordinal*

Description

Define variables as ordinal in latent variable model object

Usage

```
ordinal(x, ...) <- value
```

Arguments

x	Object
...	additional arguments to lower level functions
value	variable (formula or character vector)

Examples

```
if (requireNamespace("mets")) {
m <- lvm(y + z ~ x + 1*u[0], latent=~u)
ordinal(m, K=3) <- ~y+z
d <- sim(m, 100, seed=1)
e <- estimate(m, d)
}
```

ordreg *Univariate cumulative link regression models*

Description

Ordinal regression models

Usage

```
ordreg(
  formula,
  data = parent.frame(),
  offset,
  family = stats::binomial("probit"),
  start,
  fast = FALSE,
  ...
)
```

Arguments

formula	formula
data	data.frame
offset	offset
family	family (default proportional odds)
start	optional starting values
fast	If TRUE standard errors etc. will not be calculated
...	Additional arguments to lower level functions

Author(s)

Klaus K. Holst

Examples

```
m <- lvm(y~x)
ordinal(m,K=3) <- ~y
d <- sim(m,100)
e <- ordreg(y~x,d)
```

parpos *Generic method for finding indeces of model parameters*

Description

Generic method for finding indeces of model parameters

Usage

```
parpos(x, ...)
```

Arguments

x	Model object
...	Additional arguments

Author(s)

Klaus K. Holst

partialcor *Calculate partial correlations*

Description

Calculate partial correlation coefficients and confidence limits via Fishers z-transform

Usage

```
partialcor(formula, data, level = 0.95, ...)
```

Arguments

formula	formula speciying the covariates and optionally the outcomes to calculate partial correlation for
data	data.frame
level	Level of confidence limits
...	Additional arguments to lower level functions

Value

A coefficient matrix

Author(s)

Klaus K. Holst

Examples

```
m <- lvm(c(y1,y2,y3)~x1+x2)
covariance(m) <- c(y1,y2,y3)~y1+y2+y3
d <- sim(m,500)
partialcor(~x1+x2,d)
```

---

path *Extract pathways in model graph*

Description

Extract all possible paths from one variable to another connected component in a latent variable model. In an estimated model the effect size is decomposed into direct, indirect and total effects including approximate standard errors.

Usage

```
## S3 method for class 'lvm'
 path(object, to = NULL, from, all=FALSE, ...)
## S3 method for class 'lvmfit'
 effects(object, to, from, ...)
```

Arguments

object	Model object (lvm)
...	Additional arguments to be passed to the low level functions
to	Outcome variable (string). Alternatively a formula specifying response and predictor in which case the argument from is ignored.
from	Response variable (string), not necessarily directly affected by to.
all	If TRUE all simple paths (in undirected graph) is returned on/off.

Value

If object is of class lvmfit a list with the following elements is returned

idx	A list where each element defines a possible pathway via a integer vector indicating the index of the visited nodes.
V	A List of covariance matrices for each path.
coef	A list of parameters estimates for each path
path	A list where each element defines a possible pathway via a character vector naming the visited nodes in order.
edges	Description of 'comp2'

If object is of class lvm only the path element will be returned.

The effects method returns an object of class effects.

Note

For a `lvmfit`-object the parameters estimates and their corresponding covariance matrix are also returned. The `effects`-function additionally calculates the total and indirect effects with approximate standard errors

Author(s)

Klaus K. Holst

See Also

`children`, `parents`

Examples

```
m <- lvm(c(y1,y2,y3)~eta)
regression(m) <- y2~x1
latent(m) <- ~eta
regression(m) <- eta~x1+x2
d <- sim(m,500)
e <- estimate(m,d)

path(Model(e),y2~x1)
parents(Model(e), ~y2)
children(Model(e), ~x2)
children(Model(e), ~x2+eta)
effects(e,y2~x1)
## All simple paths (undirected)
path(m,y1~x1,all=TRUE)
```

---

pcor *Polychoric correlation*

Description

Maximum likelihood estimates of polychoric correlations

Usage

```
pcor(x, y, X, start, ...)
```

Arguments

`x`	Variable 1
`y`	Variable 2
`X`	Optional covariates
`start`	Optional starting values
`...`	Additional arguments to lower level functions

PD *Dose response calculation for binomial regression models*

Description

Dose response calculation for binomial regression models

Usage

```
PD(
  model,
  intercept = 1,
  slope = 2,
  prob = NULL,
  x,
  level = 0.5,
  ci.level = 0.95,
  vcov,
  family,
  EB = NULL
)
```

Arguments

model	Model object or vector of parameter estimates
intercept	Index of intercept parameters
slope	Index of intercept parameters
prob	Index of mixture parameters (only relevant for zibreg models)
x	Optional weights length(x)=length(intercept)+length(slope)+length(prob)
level	Probability at which level to calculate dose
ci.level	Level of confidence limits
vcov	Optional estimate of variance matrix of parameter estimates
family	Optional distributional family argument
EB	Optional ratio of treatment effect and adverse effects used to find optimal dose (regret-function argument)

Author(s)

Klaus K. Holst

pdfconvert *Convert pdf to raster format*

Description

Convert PDF file to print quality png (default 300 dpi)

Usage

```
pdfconvert(
  files,
  dpi = 300,
  resolution = 1024,
  gs,
  gsopt,
  resize,
  format = "png",
  ...
)
```

Arguments

files	Vector of (pdf-)filenames to process
dpi	DPI
resolution	Resolution of raster image file
gs	Optional ghostscript command
gsopt	Optional ghostscript arguments
resize	Optional resize arguments (mogrify)
format	Raster format (e.g. png, jpg, tif, ...)
...	Additional arguments

Details

Access to ghostscript program 'gs' is needed

Author(s)

Klaus K. Holst

See Also

dev.copy2pdf, printdev

plot.estimate *Plot method for 'estimate' objects*

Description

Plot method for 'estimate' objects

Usage

```
## S3 method for class 'estimate'
plot(
  x,
  f,
  idx,
  intercept = FALSE,
  data,
  confint = TRUE,
  type = "l",
  xlab = "x",
  ylab = "f(x)",
  col = 1,
  add = FALSE,
  null = 0,
  ...
)
```

Arguments

x	estimate object
f	function of parameter coefficients and data parsed on to 'estimate'. If omitted a forest-plot will be produced.
idx	Index of parameters (default all)
intercept	include intercept in forest-plot
data	data.frame
confint	Add confidence limits
type	plot type ('l')
xlab	x-axis label
ylab	y-axis label
col	color
add	add plot to current device
null	null value for forest-plot
...	additional arguments to lower-level functions

plot.lvm *Plot path diagram*

Description

Plot the path diagram of a SEM

Usage

```
## S3 method for class 'lvm'
plot(
  x,
  diag = FALSE,
  cor = TRUE,
  labels = FALSE,
  intercept = FALSE,
  addcolor = TRUE,
  plain = FALSE,
  cex,
  fontsize1 = 10,
  noplot = FALSE,
  graph = list(rankdir = "BT"),
  attrs = list(graph = graph),
  unexpr = FALSE,
  addstyle = TRUE,
  plot.engine = lava.options()$plot.engine,
  init = TRUE,
  layout = lava.options()$layout,
  edgecolor = lava.options()$edgecolor,
  graph.proc = lava.options()$graph.proc,
  ...
)
```

Arguments

x	Model object
diag	Logical argument indicating whether to visualize variance parameters (i.e. diagonal of variance matrix)
cor	Logical argument indicating whether to visualize correlation parameters
labels	Logical argument indiciating whether to add labels to plot (Unnamed parameters will be labeled p1,p2,...)
intercept	Logical argument indiciating whether to add intercept labels
addcolor	Logical argument indiciating whether to add colors to plot (overrides nodecolor calls)
plain	if TRUE strip plot of colors and boxes

cex	Fontsize of node labels
fontsize1	Fontsize of edge labels
noplot	if TRUE then return graphNEL object only
graph	Graph attributes (Rgraphviz)
attrs	Attributes (Rgraphviz)
unexpr	if TRUE remove expressions from labels
addstyle	Logical argument indicating whether additional style should automatically be added to the plot (e.g. dashed lines to double-headed arrows)
plot.engine	default 'Rgraphviz' if available, otherwise visNetwork,igraph
init	Reinitialize graph (for internal use)
layout	Graph layout (see Rgraphviz or igraph manual)
edgecolor	if TRUE plot style with colored edges
graph.proc	Function that post-process the graph object (default: subscripts are automatically added to labels of the nodes)
...	Additional arguments to be passed to the low level functions

Author(s)

Klaus K. Holst

Examples

```
if (interactive()) {
m <- lvm(c(y1,y2) ~ eta)
regression(m) <- eta ~ z+x2
regression(m) <- c(eta,z) ~ x1
latent(m) <- ~eta
labels(m) <- c(y1=expression(y[scriptscriptstyle(1)]),
y2=expression(y[scriptscriptstyle(2)]),
x1=expression(x[scriptscriptstyle(1)]),
x2=expression(x[scriptscriptstyle(2)]),
eta=expression(eta))
edgelabels(m, eta ~ z+x1+x2, cex=2, lwd=3,
           col=c("orange","lightblue","lightblue")) <- expression(rho,phi,psi)
nodecolor(m, vars(m), border="white", labcol="darkblue") <- NA
nodecolor(m, ~y1+y2+z, labcol=c("white","white","black")) <- NA
plot(m,cex=1.5)

d <- sim(m,100)
e <- estimate(m,d)
plot(e)

m <- lvm(c(y1,y2) ~ eta)
regression(m) <- eta ~ z+x2
regression(m) <- c(eta,z) ~ x1
latent(m) <- ~eta
plot(lava:::beautify(m,edgecol=FALSE))
}
```

plot.sim *Plot method for simulation 'sim' objects*

Description

Density and scatter plots

Usage

```
## S3 method for class 'sim'
plot(
  x,
  estimate,
  se = NULL,
  true = NULL,
  names = NULL,
  auto.layout = TRUE,
  byrow = FALSE,
  type = "p",
  ask = grDevices::dev.interactive(),
  col = c("gray60", "orange", "darkblue", "seagreen", "darkred"),
  pch = 16,
  cex = 0.5,
  lty = 1,
  lwd = 0.3,
  legend,
  legendpos = "topleft",
  cex.legend = 0.8,
  plot.type = c("multiple", "single"),
  polygon = TRUE,
  density = NULL,
  angle = -45,
  cex.axis = 0.8,
  alpha = 0.2,
  main,
  cex.main = 1,
  equal = FALSE,
  delta = 1.15,
  ylim = NULL,
  xlim = NULL,
  ylab = "",
  xlab = "",
  rug = FALSE,
  rug.alpha = 0.5,
  line.col = scatter.col,
  line.lwd = 1,
  line.lty = 1,
```

```
  line.alpha = 1,
  scatter.ylab = "Estimate",
  scatter.ylim = NULL,
  scatter.xlim = NULL,
  scatter.alpha = 0.5,
  scatter.col = col,
  border = col,
  true.lty = 2,
  true.col = "gray70",
  true.lwd = 1.2,
  density.plot = TRUE,
  scatter.plot = FALSE,
  running.mean = scatter.plot,
  add = FALSE,
  ...
)
```

Arguments

x	sim object
estimate	columns with estimates
se	columns with standard error estimates
true	(optional) vector of true parameter values
names	(optional) names of estimates
auto.layout	Auto layout (default TRUE)
byrow	Add new plots to layout by row
type	plot type
ask	if TRUE user is asked for input, before a new figure is drawn
col	colour (for each estimate)
pch	plot symbol
cex	point size
lty	line type
lwd	line width
legend	legend
legendpos	legend position
cex.legend	size of legend text
plot.type	'single' or 'multiple' (default)
polygon	if TRUE fill the density estimates with colour
density	if non-zero add shading lines to polygon
angle	shading lines angle of polygon
cex.axis	Font size on axis
alpha	Semi-transparent level (1: non-transparent, 0: full)

main	Main title
cex.main	Size of title font
equal	Same x-axis and y-axis for all plots
delta	Controls the amount of space around axis limits
ylim	y-axis limits
xlim	x-axis limits
ylab	y axis label
xlab	x axis label
rug	if TRUE add rug representation of data to x-axis
rug.alpha	rug semi-transparency level
line.col	line colour (running mean, only for scatter plots)
line.lwd	line width (running mean, only for scatter plots)
line.lty	line type (running mean, only for scatter plots)
line.alpha	line transparency
scatter.ylab	y label for density plots
scatter.ylim	y-axis limits for density plots
scatter.xlim	x-axis limits for density plots
scatter.alpha	semi-transparency of scatter plot
scatter.col	scatter plot colour
border	border colour of density estimates
true.lty	true parameter estimate line type
true.col	true parameter colour
true.lwd	true parameter line width
density.plot	if TRUE add density plot
scatter.plot	if TRUE add scatter plot
running.mean	if TRUE add running average estimate to scatter plot
add	if TRUE add to existing plot
...	additional arguments to lower level functions

Examples

```
n <- 1000
val <- cbind(est1=rnorm(n,sd=1),est2=rnorm(n,sd=0.2),est3=rnorm(n,1,sd=0.5),
             sd1=runif(n,0.8,1.2),sd2=runif(n,0.1,0.3),sd3=runif(n,0.25,0.75))

plot.sim(val,estimate=c(1,2),true=c(0,0),se=c(4,5),equal=TRUE,scatter.plot=TRUE)
plot.sim(val,estimate=c(1,3),true=c(0,1),se=c(4,6),xlim=c(-3,3),
scatter.ylim=c(-3,3),scatter.plot=TRUE)
plot.sim(val,estimate=c(1,2),true=c(0,0),se=c(4,5),equal=TRUE,
plot.type="single",scatter.plot=TRUE)
plot.sim(val,estimate=c(1),se=c(4,5,6),plot.type="single",scatter.plot=TRUE)
```

```
plot.sim(val,estimate=c(1,2,3),equal=TRUE,scatter.plot=TRUE)
plot.sim(val,estimate=c(1,2,3),equal=TRUE,byrow=TRUE,scatter.plot=TRUE)
plot.sim(val,estimate=c(1,2,3),plot.type="single",scatter.plot=TRUE)
plot.sim(val,estimate=1,se=c(3,4,5),plot.type="single",scatter.plot=TRUE)

density.sim(val,estimate=c(1,2,3),density=c(0,10,10), lwd=2, angle=c(0,45,-45),cex.legend=1.3)
```

---

plotConf *Plot regression lines*

Description

Plot regression line (with interactions) and partial residuals.

Usage

```
plotConf(
  model,
  var1 = NULL,
  var2 = NULL,
  data = NULL,
  ci.lty = 0,
  ci = TRUE,
  level = 0.95,
  pch = 16,
  lty = 1,
  lwd = 2,
  npoints = 100,
  xlim,
  col = NULL,
  colpt,
  alpha = 0.5,
  cex = 1,
  delta = 0.07,
  centermark = 0.03,
  jitter = 0.2,
  cidiff = FALSE,
  mean = TRUE,
  legend = ifelse(is.null(var1), FALSE, "topright"),
  trans = function(x) {
     x
 },
  partres = inherits(model, "lm"),
  partse = FALSE,
  labels,
  vcov,
  predictfun,
```

```
  plot = TRUE,
  new = TRUE,
  ...
)
```

Arguments

model	Model object (e.g. lm())
var1	predictor (Continuous or factor)
var2	Factor that interacts with var1
data	data.frame to use for prediction (model.frame is used as default)
ci.lty	Line type for confidence limits
ci	Boolean indicating wether to draw pointwise 95% confidence limits
level	Level of confidence limits (default 95%)
pch	Point type for partial residuals
lty	Line type for estimated regression lines
lwd	Line width for regression lines
npoints	Number of points used to plot curves
xlim	Range of x axis
col	Color (for each level in var2)
colpt	Color of partial residual points
alpha	Alpha level
cex	Point size
delta	For categorical var1
centermark	For categorical var1
jitter	For categorical var1
cidiff	For categorical var1
mean	For categorical var1
legend	Boolean (add legend)
trans	Transform estimates (e.g. exponential)
partres	Boolean indicating whether to plot partial residuals
partse	.
labels	Optional labels of var2
vcov	Optional variance estimates
predictfun	Optional predict-function used to calculate confidence limits and predictions
plot	If FALSE return only predictions and confidence bands
new	If FALSE add to current plot
...	additional arguments to lower level functions

Value

list with following members:

- x Variable on the x-axis var1
- y Variable on the y-axis (partial residuals)
- predict Matrix with confidence limits and predicted startvalues.R

Author(s)

Klaus K. Holst

See Also

termplot()

Examples

```
n <- 100
x0 <- rnorm(n)
x1 <- seq(-3,3, length.out=n)
x2 <- factor(rep(c(1,2),each=n/2), labels=c("A","B"))
y <- 5 + 2*x0 + 0.5*x1 + -1*(x2=="B")*x1 + 0.5*(x2=="B") + rnorm(n, sd=0.25)
dd <- data.frame(y=y, x1=x1, x2=x2)
lm0 <- lm(y ~ x0 + x1*x2, dd)
plotConf(lm0, var1="x1", var2="x2")
abline(a=5,b=0.5,col="red")
abline(a=5.5,b=-0.5,col="red")
## points(5+0.5*x1 -1*(x2=="B")*x1 + 0.5*(x2=="B") ~ x1, cex=2)

data(iris)
l <- lm(Sepal.Length ~ Sepal.Width*Species,iris)
plotConf(l,var2="Species")
plotConf(l,var1="Sepal.Width",var2="Species")

## Not run:
## lme4 model
dd$Id <- rbinom(n, size = 3, prob = 0.3)
lmer0 <- lme4::lmer(y ~ x0 + x1*x2 + (1|Id), dd)
plotConf(lmer0, var1="x1", var2="x2")

## End(Not run)
```

---

predict.lvm *Prediction in structural equation models*

Description

Prediction in structural equation models

Usage

```
## S3 method for class 'lvm'
predict(
  object,
  x = NULL,
  y = NULL,
  residual = FALSE,
  p,
  data,
  path = FALSE,
  quick = is.null(x) & !(residual | path),
  ...
)
```

Arguments

object	Model object
x	optional list of (endogenous) variables to condition on
y	optional subset of variables to predict
residual	If true the residuals are predicted
p	Parameter vector
data	Data to use in prediction
path	Path prediction
quick	If TRUE the conditional mean and variance given covariates are returned (and all other calculations skipped)
...	Additional arguments to lower level function

See Also

predictlvm

Examples

```
m <- lvm(list(c(y1,y2,y3)~u,u~x)); latent(m) <- ~u
d <- sim(m,100)
e <- estimate(m,d)

## Conditional mean (and variance as attribute) given covariates
r <- predict(e)
## Best linear unbiased predictor (BLUP)
r <- predict(e,vars(e))
##  Conditional mean of y3 giving covariates and y1,y2
r <- predict(e,y3~y1+y2)
##  Conditional mean  gives covariates and y1
r <- predict(e,~y1)
##  Predicted residuals (conditional on all observed variables)
r <- predict(e,vars(e),residual=TRUE)
```

predictlvm *Predict function for latent variable models*

Description

Predictions of conditinoal mean and variance and calculation of jacobian with respect to parameter vector.

Usage

```
predictlvm(object, formula, p = coef(object), data = model.frame(object), ...)
```

Arguments

object	Model object
formula	Formula specifying which variables to predict and which to condition on
p	Parameter vector
data	Data.frame
...	Additional arguments to lower level functions

See Also

predict.lvm

Examples

```
m <- lvm(c(x1,x2,x3)~u1,u1~z,
         c(y1,y2,y3)~u2,u2~u1+z)
latent(m) <- ~u1+u2
d <- simulate(m,10,"u2,u2"=2,"u1,u1"=0.5,seed=123)
e <- estimate(m,d)

## Conditional mean given covariates
predictlvm(e,c(x1,x2)~1)$mean
## Conditional variance of u1,y1 given x1,x2
predictlvm(e,c(u1,y1)~x1+x2)$var
```

predict_glm *Predict from a GLM with modified coefficients*

Description

Compute predictions from a fitted stats::glm object, optionally substituting new parameter values. Unlike stats::predict.glm, this function allows the user to supply an arbitrary coefficient vector `p`, which is useful for computing predictions at counterfactual parameter values (e.g., during optimization or simulation). The returned value also includes a `"grad"` attribute containing the Jacobian of predictions with respect to the coefficients.

Usage

```
predict_glm(
  object,
  p = coef(object),
  data,
  offset = NULL,
  type = c("response", "link"),
  ...
)
```

Arguments

`object`	A fitted `glm` object.
`p`	Numeric vector of coefficients (defaults to `coef(x)`).
`data`	Optional data frame for computing the model matrix and response. If missing, the original model matrix from the fitted object is used.
`offset`	Optional offset vector. If `NULL` (default), the offset stored in the fitted model (`x$offset`) is used.
`type`	Character; `"response"` (default) returns predictions on the response scale via the inverse link function, `"link"` returns predictions on the linear predictor scale.
`...`	Additional arguments (currently unused).

Value

Numeric vector of predictions with a `"grad"` attribute containing the gradient (Jacobian) of predictions with respect to the coefficients. When `type = "link"`, the gradient is simply the model matrix `X`.

See Also

stats::predict.glm, score.glm

Examples

```
m <- glm(mpg ~ hp + wt, data = mtcars)
p0 <- coef(m)
# Predictions at fitted coefficients match predict.glm
all.equal(as.numeric(predict_glm(m)), fitted(m))
# Predictions with modified coefficients
predict_glm(m, p = p0 * 1.1)
```

Print *Generic print method*

Description

Nicer print method for tabular data. Falls back to standard print method for all other data types.

Usage

```
Print(x, n = 5, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

x	object to print
n	number of rows to show from top and bottom of tabular data
digits	precision
...	additional arguments to print method

Range.lvm *Define range constraints of parameters*

Description

Define range constraints of parameters

Usage

```
Range.lvm(a = 0, b = 1)
```

Arguments

a	Lower bound
b	Upper bound

Value

function

Author(s)

Klaus K. Holst

`rbind.Surv` *Appending* `Surv` *objects*

Description

`rbind` method for `Surv` objects

Usage

```
## S3 method for class 'Surv'
rbind(...)
```

Arguments

`...`	`Surv` objects

Value

`Surv` object

Author(s)

Klaus K. Holst

Examples

```
y <- yl <- yr <- rnorm(10)
yl[1:5] <- NA; yr[6:10] <- NA
S1 <- survival::Surv(yl,yr,type="interval2")
S2 <- survival::Surv(y,y>0,type="right")
S3 <- survival::Surv(y,y<0,type="left")

rbind(S1,S1)
rbind(S2,S2)
rbind(S3,S3)
```

regression<- *Add regression association to latent variable model*

Description

Define regression association between variables in a lvm-object and define linear constraints between model equations.

Usage

```
## S3 method for class 'lvm'
regression(object = lvm(), to, from, fn = NA,
messages = lava.options()$messages, additive=TRUE, y, x, value, ...)
## S3 replacement method for class 'lvm'
regression(object, to=NULL, from=NULL, ...) <- value
```

Arguments

object	lvm-object.
...	Additional arguments to be passed to the low level functions
value	A formula specifying the linear constraints or if to=NULL a list of parameter values.
to	Character vector of outcome(s) or formula object.
from	Character vector of predictor(s).
fn	Real function defining the functional form of predictors (for simulation only).
messages	Controls which messages are turned on/off (0: all off)
additive	If FALSE and predictor is categorical a non-additive effect is assumed
y	Alias for 'to'
x	Alias for 'from'

Details

The regression function is used to specify linear associations between variables of a latent variable model, and offers formula syntax resembling the model specification of e.g. lm.

For instance, to add the following linear regression model, to the lvm-object, m:

$$E(Y|X_1, X_2) = \beta_1 X_1 + \beta_2 X_2$$

We can write

```
regression(m) <- y ~ x1 + x2
```

Multivariate models can be specified by successive calls with regression, but multivariate formulas are also supported, e.g.

```
regression(m) <- c(y1,y2) ~ x1 + x2
```

defines

$$E(Y_i|X_1, X_2) = \beta_{1i}X_1 + \beta_{2i}X_2$$

The special function, `f`, can be used in the model specification to specify linear constraints. E.g. to fix $\beta_1 = \beta_2$, we could write

```
regression(m) <- y ~ f(x1,beta) + f(x2,beta)
```

The second argument of `f` can also be a number (e.g. defining an offset) or be set to `NA` in order to clear any previously defined linear constraints.

Alternatively, a more straight forward notation can be used:

```
regression(m) <- y ~ beta*x1 + beta*x2
```

All the parameter values of the linear constraints can be given as the right handside expression of the assigment function `regression<-` if the first (and possibly second) argument is defined as well. E.g:

```
regression(m,y1~x1+x2) <- list("a1","b1")
```

defines $E(Y_1|X_1, X_2) = a1X_1 + b1X_2$. The rhs argument can be a mixture of character and numeric values (and NA's to remove constraints).

The function `regression` (called without additional arguments) can be used to inspect the linear constraints of a `lvm`-object.

Value

A `lvm`-object

Note

Variables will be added to the model if not already present.

Author(s)

Klaus K. Holst

See Also

intercept<-, covariance<-, constrain<-, parameter<-, latent<-, cancel<-, kill<-

Examples

```
m <- lvm() ## Initialize empty lvm-object
## E(y1|z,v) = beta1*z + beta2*v
regression(m) <- y1 ~ z + v
## E(y2|x,z,v) = beta*x + beta*z + 2*v + beta3*u
regression(m) <- y2 ~ f(x,beta) + f(z,beta)  + f(v,2) + u
## Clear restriction on association between y and
## fix slope coefficient of u to beta
regression(m, y2 ~ v+u) <- list(NA,"beta")

regression(m) ## Examine current linear parameter constraints
```

```
## A multivariate model, E(yi|x1,x2) = beta[1i]*x1 + beta[2i]*x2:
m2 <- lvm(c(y1,y2) ~ x1+x2)
```

`revdiag` *Create/extract 'reverse'-diagonal matrix or off-diagonal elements*

Description

Create/extract 'reverse'-diagonal matrix or off-diagonal elements

Usage

```
revdiag(x,...)
offdiag(x,type=0,...)

revdiag(x,...) <- value
offdiag(x,type=0,...) <- value
```

Arguments

`x`	vector
`...`	additional arguments to lower level functions
`value`	For the assignment function the values to put in the diagonal
`type`	0: upper and lower triangular, 1: upper triangular, 2: lower triangular, 3: upper triangular + diagonal, 4: lower triangular + diagonal

Author(s)

Klaus K. Holst

`rmvar` *Remove variables from (model) object.*

Description

Generic method for removing elements of object

Usage

```
rmvar(x, ...) <- value
```

Arguments

x	Model object
...	additional arguments to lower level functions
value	Vector of variables or formula specifying which nodes to remove

Author(s)

Klaus K. Holst

See Also

cancel

Examples

```
m <- lvm()
addvar(m) <- ~y1+y2+x
covariance(m) <- y1~y2
regression(m) <- c(y1,y2) ~ x
## Cancel the covariance between the residuals of y1 and y2
cancel(m) <- y1~y2
## Remove y2 from the model
rmvar(m) <- ~y2
```

---

rotate2 *Performs a rotation in the plane*

Description

Performs a rotation in the plane

Usage

```
rotate2(x, theta = pi)
```

Arguments

x	Matrix to be rotated (2 times n)
theta	Rotation in radians

Value

Returns a matrix of the same dimension as x

Author(s)

Klaus K. Holst

Examples

```
rotate2(cbind(c(1,2),c(2,1)))
```

---

scheffe *Calculate simultaneous confidence limits by Scheffe's method*

Description

Function to compute the Scheffe corrected confidence interval for the regression line

Usage

```
scheffe(model, newdata = model.frame(model), level = 0.95)
```

Arguments

model	Linear model
newdata	new data frame
level	confidence level (0.95)

Examples

```
x <- rnorm(100)
d <- data.frame(y=rnorm(length(x),x),x=x)
l <- lm(y~x,d)
plot(y~x,d)
abline(l)
d0 <- data.frame(x=seq(-5,5,length.out=100))
d1 <- cbind(d0,predict(l,newdata=d0,interval="confidence"))
d2 <- cbind(d0,scheffe(l,d0))
lines(lwr~x,d1,lty=2,col="red")
lines(upr~x,d1,lty=2,col="red")
lines(lwr~x,d2,lty=2,col="blue")
lines(upr~x,d2,lty=2,col="blue")
```

---

score *Extract score function*

Description

Extract the score function (gradient of the log-likelihood) from a model object.

Usage

```
score(x, ...)
```

Arguments

x	Model object
...	Additional arguments to lower level functions

Value

Matrix of score/gradient values. Rows correspond to observations (if `indiv=TRUE`) or a single row, columns to parameters.

---

semdata *Example SEM data*

Description

Simulated data

Format

data.frame

Source

Simulated

---

serotonin *Serotonin data*

Description

This simulated data mimics a PET imaging study where the 5-HT2A receptor and serotonin transporter (SERT) binding potential has been quantified into 8 different regions. The 5-HT2A cortical regions are considered high-binding regions measurements. These measurements can be regarded as proxy measures of the extra-cellular levels of serotonin in the brain

day	numeric	Scan day of the year
age	numeric	Age at baseline scan
mem	numeric	Memory performance score
depr	numeric	Depression (mild) status 500 days after baseline
gene1	numeric	Gene marker 1 (HTR2A)
gene2	numeric	Gene marker 2 (HTTTLPR)
cau	numeric	SERT binding, Caudate Nucleus
th	numeric	SERT binding, Thalamus
put	numeric	SERT binding, Putamen
mid	numeric	SERT binding, Midbrain

aci	numeric	5-HT2A binding, Anterior cingulate gyrus
pci	numeric	5-HT2A binding, Posterior cingulate gyrus
sfc	numeric	5-HT2A binding, Superior frontal cortex
par	numeric	5-HT2A binding, Parietal cortex

Format

data.frame

Source

Simulated

`sim.default` *Monte Carlo simulation*

Description

Applies a function repeatedly for a specified number of replications or over a list/data.frame with plot and summary methods for summarizing the Monte Carlo experiment. Can be parallelized via the future package (use the `future::plan()` function).

Usage

```
## Default S3 method:
sim(
  x = NULL,
  R = 100,
  f = NULL,
  colnames = NULL,
  seed = NULL,
  args = list(),
  iter = FALSE,
  mc.cores,
  progressr.message = NULL,
  estimate.index = 1:2,
  ...
)
```

Arguments

`x`	function or 'sim' object
`R`	Number of replications or data.frame with parameters
`f`	Optional function (i.e., if x is a matrix)
`colnames`	Optional column names

seed	(optional) Seed (needed with cl=TRUE)
args	(optional) list of named arguments passed to (mc)mapply
iter	If TRUE the iteration number is passed as first argument to (mc)mapply
mc.cores	Optional number of cores. Will use parallel::mcmapply instead of future
progressr.message	Optional message for the progressr progress-bar
estimate.index	If return object inherits from estimate then only these column indices are extracted (estimate, se, lower, upper, p-val)
...	Additional arguments to future.apply::future_mapply()

Details

To parallelize the calculation use the future::plan() function (e.g., future::plan(multisession()) to distribute the calculations over the R replications on all available cores). The output is controlled via the progressr package (e.g., progressr::handlers(global=TRUE) to enable progress information).

See Also

summary.sim() plot.sim() sim.lvm()

Examples

```
m <- lvm(y~x+e)
distribution(m,~y) <- 0
distribution(m,~x) <- uniform.lvm(a=-1.1,b=1.1)
transform(m,e~x) <- function(x) (1*x^4)*rnorm(length(x),sd=1)

onerun <- function(iter=NULL,...,n=2e3,b0=1,idx=2) {
    d <- sim(m,n,p=c("y~x"=b0))
    l <- lm(y~x,d)
    res <- c(coef(summary(l))[idx,1:2],
             confint(l)[idx,],
             coef(estimate(l), mat=TRUE)[idx,2:4])
    names(res) <- c("Estimate","Model.se","Model.lo","Model.hi",
                    "Sandwich.se","Sandwich.lo","Sandwich.hi")
    res
}
val <- sim(onerun,R=10,b0=1)
val

val <- sim(val,R=40,b0=1) ## append results
summary(val,estimate=c(1,1),confint=c(3,4,6,7),true=c(1,1))

summary(val,estimate=c(1,1),se=c(2,5),names=c("Model","Sandwich"))
summary(val,estimate=c(1,1),se=c(2,5),true=c(1,1),
        names=c("Model","Sandwich"),confint=TRUE)

if (interactive()) {
    plot(val,estimate=1,c(2,5),true=1,
```

```
          names=c("Model","Sandwich"),polygon=FALSE)
    plot(val,estimate=c(1,1),se=c(2,5),main=NULL,
          true=c(1,1),names=c("Model","Sandwich"),
          line.lwd=1,col=c("gray20","gray60"),
          rug=FALSE)
    plot(val,estimate=c(1,1),se=c(2,5),true=c(1,1),
          names=c("Model","Sandwich"))
}

f <- function(a=1, b=1) {
  rep(a*b, 5)
}
R <- Expand(a=1:3, b=1:3)
sim(f, R)
sim(function(a,b) f(a,b), 3, args=c(a=5,b=5))
sim(function(iter=1,a=5,b=5) iter*f(a,b), iter=TRUE, R=5)

## Returning estimate objects with extra per-iteration information
onerun2 <- function(...) {
  x <- rnorm(50)
  y <- 0.5*x + rnorm(50)
  e <- estimate(lm(y ~ x))
  c(e, converged = 1, niter = sample(5:20, 1), drop.ic = TRUE)
  # works identically with summary.estimate objects
  # c(summary(e), converged = 1, niter = sample(5:20, 1), drop.ic = TRUE)
}
val2 <- sim(onerun2, R = 10)
val2
# extra columns are stored but excluded from default summary statistics
idx <- c("(Intercept)", "x", "niter")
summary(val2, estimate=idx, true=c(0,0.5,NA))
```

`sim.lvm` *Simulate model*

Description

Simulate data from a general SEM model including non-linear effects and general link and distribution of variables.

Usage

```
## S3 method for class 'lvm'
sim(x, n = NULL, p = NULL, normal = FALSE, cond = FALSE,
sigma = 1, rho = 0.5, X = NULL, unlink=FALSE, latent=TRUE,
use.labels = TRUE, seed=NULL, ...)
```

Arguments

x	Model object
n	Number of simulated values/individuals
p	Parameter value (optional)
normal	Logical indicating whether to simulate data from a multivariate normal distribution conditional on exogenous variables hence ignoring functional/distribution definition
cond	for internal use
sigma	Default residual variance (1)
rho	Default covariance parameter (0.5)
X	Optional matrix of fixed values of variables (manipulation)
unlink	Return Inverse link transformed data
latent	Include latent variables (default TRUE)
use.labels	convert categorical variables to factors before applying transformation
seed	Random seed
...	Additional arguments to be passed to the low level functions

Author(s)

Klaus K. Holst

Examples

```
##################################################
## Logistic regression
##################################################
m <- lvm(y~x+z)
regression(m) <- x~z
distribution(m,~y+z) <- binomial.lvm("logit")
d <- sim(m,1e3)
head(d)

e <- estimate(m,d,estimator="glm")
e
## Simulate a few observation from estimated model
sim(e,n=5)

##################################################
## Poisson
##################################################
distribution(m,~y) <- poisson.lvm()
d <- sim(m,1e4,p=c(y=-1,"y~x"=2,z=1))
head(d)
estimate(m,d,estimator="glm")
mean(d$z); lava:::expit(1)

summary(lm(y~x,sim(lvm(y[1:2]~4*x),1e3)))
```

```
##################################################
### Gamma distribution
##################################################
m <- lvm(y~x)
distribution(m,~y+x) <- list(Gamma.lvm(shape=2),binomial.lvm())
intercept(m,~y) <- 0.5
d <- sim(m,1e4)
summary(g <- glm(y~x,family=Gamma(),data=d))
## Not run: MASS::gamma.shape(g)

args(lava::Gamma.lvm)
distribution(m,~y) <- Gamma.lvm(shape=2,log=TRUE)
sim(m,10,p=c(y=0.5))[,"y"]

##################################################
### Beta
##################################################
m <- lvm()
distribution(m,~y) <- beta.lvm(alpha=2,beta=1)
var(sim(m,100,"y,y"=2))
distribution(m,~y) <- beta.lvm(alpha=2,beta=1,scale=FALSE)
var(sim(m,100))

##################################################
### Transform
##################################################
m <- lvm()
transform(m,xz~x+z) <- function(x) x[1]*(x[2]>0)
regression(m) <- y~x+z+xz
d <- sim(m,1e3)
summary(lm(y~x+z + x*I(z>0),d))

##################################################
### Non-random variables
##################################################
m <- lvm()
distribution(m,~x+z+v+w) <- list(Sequence.lvm(0,5),## Seq. 0 to 5 by 1/n
                                 Binary.lvm(),       ## Vector of ones
                                 Binary.lvm(0.5),    ##  0.5n 0, 0.5n 1
                                 Binary.lvm(interval=list(c(0.3,0.5),c(0.8,1))))
sim(m,10)

##################################################
### Cox model
### piecewise constant hazard
##################################################
m <- lvm(t~x)
rates <- c(1,0.5); cuts <- c(0,5)
## Constant rate: 1 in [0,5), 0.5 in [5,Inf)
distribution(m,~t) <- coxExponential.lvm(rate=rates,timecut=cuts)

## Not run:
```

```
    d <- sim(m,2e4,p=c("t~x"=0.1)); d$status <- TRUE
    plot(timereg::aalen(survival::Surv(t,status)~x,data=d,
                        resample.iid=0,robust=0),spec=1)
    L <- approxfun(c(cuts,max(d$t)),f=1,
                   cumsum(c(0,rates*diff(c(cuts,max(d$t))))),
                   method="linear")
    curve(L,0,100,add=TRUE,col="blue")

## End(Not run)

################################################
### Cox model
### piecewise constant hazard, gamma frailty
################################################
m <- lvm(y~x+z)
rates <- c(0.3,0.5); cuts <- c(0,5)
distribution(m,~y+z) <- list(coxExponential.lvm(rate=rates,timecut=cuts),
                             loggamma.lvm(rate=1,shape=1))
## Not run:
    d <- sim(m,2e4,p=c("y~x"=0,"y~z"=0)); d$status <- TRUE
    plot(timereg::aalen(survival::Surv(y,status)~x,data=d,
                        resample.iid=0,robust=0),spec=1)
    L <- approxfun(c(cuts,max(d$y)),f=1,
                   cumsum(c(0,rates*diff(c(cuts,max(d$y))))),
                   method="linear")
    curve(L,0,100,add=TRUE,col="blue")

## End(Not run)
## Equivalent via transform (here with Aalens additive hazard model)
m <- lvm(y~x)
distribution(m,~y) <- aalenExponential.lvm(rate=rates,timecut=cuts)
distribution(m,~z) <- Gamma.lvm(rate=1,shape=1)
transform(m,t~y+z) <- prod
sim(m,10)
## Shared frailty
m <- lvm(c(t1,t2)~x+z)
rates <- c(1,0.5); cuts <- c(0,5)
distribution(m,~y) <- aalenExponential.lvm(rate=rates,timecut=cuts)
distribution(m,~z) <- loggamma.lvm(rate=1,shape=1)
## Not run:
mets::fast.reshape(sim(m,100),varying="t")

## End(Not run)

################################################
### General multivariate distributions
################################################
## Not run:
m <- lvm()
distribution(m,~y1+y2,oratio=4) <- VGAM::rbiplackcop
ksmooth2(sim(m,1e4),rgl=FALSE,theta=-20,phi=25)

m <- lvm()
```

```
distribution(m,~z1+z2,"or1") <- VGAM::rbiplackcop
distribution(m,~y1+y2,"or2") <- VGAM::rbiplackcop
sim(m,10,p=c(or1=0.1,or2=4))

## End(Not run)

m <- lvm()
distribution(m,~y1+y2+y3,TRUE) <- function(n,...) rmvn0(n,sigma=diag(3)+1)
var(sim(m,100))

## Syntax also useful for univariate generators, e.g.
m <- lvm(y~x+z)
distribution(m,~y,TRUE) <- function(n) rnorm(n,mean=1000)
sim(m,5)
distribution(m,~y,"m1",0) <- rnorm
sim(m,5)
sim(m,5,p=c(m1=100))

################################################
### Regression design in other parameters
################################################
## Variance heterogeneity
m <- lvm(y~x)
distribution(m,~y) <- function(n,mean,x) rnorm(n,mean,exp(x)^.5)
if (interactive()) plot(y~x,sim(m,1e3))
## Alternaively, calculate the standard error directly
addvar(m) <- ~sd ## If 'sd' should be part of the resulting data.frame
constrain(m,sd~x) <- function(x) exp(x)^.5
distribution(m,~y) <- function(n,mean,sd) rnorm(n,mean,sd)
if (interactive()) plot(y~x,sim(m,1e3))

## Regression on variance parameter
m <- lvm()
regression(m) <- y~x
regression(m) <- v~x
##distribution(m,~v) <- 0 # No stochastic term
## Alternative:
## regression(m) <- v[NA:0]~x
distribution(m,~y) <- function(n,mean,v) rnorm(n,mean,exp(v)^.5)
if (interactive()) plot(y~x,sim(m,1e3))

## Regression on shape parameter in Weibull model
m <- lvm()
regression(m) <- y ~ z+v
regression(m) <- s ~ exp(0.6*x-0.5*z)
distribution(m,~x+z) <- binomial.lvm()
distribution(m,~cens) <- coxWeibull.lvm(scale=1)
distribution(m,~y) <- coxWeibull.lvm(scale=0.1,shape=~s)
eventTime(m) <- time ~ min(y=1,cens=0)

if (interactive()) {
    d <- sim(m,1e3)
    require(survival)
```

```
    (cc <- coxph(Surv(time,status)~v+strata(x,z),data=d))
    plot(survfit(cc) ,col=1:4,mark.time=FALSE)
}

##################################################
### Categorical predictor
##################################################
m <- lvm()
## categorical(m,K=3) <- "v"
categorical(m,labels=c("A","B","C")) <- "v"

regression(m,additive=FALSE) <- y~v
## Not run:
plot(y~v,sim(m,1000,p=c("y~v:2"=3)))

## End(Not run)

m <- lvm()
categorical(m,labels=c("A","B","C"),p=c(0.5,0.3)) <- "v"
regression(m,additive=FALSE,beta=c(0,2,-1)) <- y~v
## equivalent to:
## regression(m,y~v,additive=FALSE) <- c(0,2,-1)
regression(m,additive=FALSE,beta=c(0,4,-1)) <- z~v
table(sim(m,1e4)$v)
glm(y~v, data=sim(m,1e4))
glm(y~v, data=sim(m,1e4,p=c("y~v:1"=3)))

transform(m,v2~v) <- function(x) x=='A'
sim(m,10)

##################################################
### Pre-calculate object
##################################################
m <- lvm(y~x)
m2 <- sim(m,'y~x'=2)
sim(m,10,'y~x'=2)
sim(m2,10) ## Faster
```

---

spaghetti *Spaghetti plot*

Description

Spaghetti plot for longitudinal data

Usage

```
spaghetti(
  formula,
  data = NULL,
```

```
  id = "id",
  group = NULL,
  type = "o",
  lty = 1,
  pch = NA,
  col = 1:10,
  alpha = 0.3,
  lwd = 1,
  level = 0.95,
  trend.formula = formula,
  tau = NULL,
  trend.lty = 1,
  trend.join = TRUE,
  trend.delta = 0.2,
  trend = !is.null(tau),
  trend.col = col,
  trend.alpha = 0.2,
  trend.lwd = 3,
  trend.jitter = 0,
  legend = NULL,
  by = NULL,
  xlab = "Time",
  ylab = "",
  add = FALSE,
  ...
)
```

Arguments

formula	Formula (response ~ time)
data	data.frame
id	Id variable
group	group variable
type	Type (line 'l', stair 's', ...)
lty	Line type
pch	Colour
col	Colour
alpha	transparency (0-1)
lwd	Line width
level	Confidence level
trend.formula	Formula for trendline
tau	Quantile to estimate (trend)
trend.lty	Trend line type
trend.join	Trend polygon

trend.delta	Length of limit bars
trend	Add trend line
trend.col	Colour of trend line
trend.alpha	Transparency
trend.lwd	Trend line width
trend.jitter	Jitter amount
legend	Legend
by	make separate plot for each level in 'by' (formula, name of column, or vector)
xlab	Label of X-axis
ylab	Label of Y-axis
add	Add to existing device
...	Additional arguments to lower level arguments

Author(s)

Klaus K. Holst

Examples

```
if (interactive() & requireNamespace("mets")) {
K <- 5
y <- "y"%++%seq(K)
m <- lvm()
regression(m,y=y,x=~u) <- 1
regression(m,y=y,x=~s) <- seq(K)-1
regression(m,y=y,x=~x) <- "b"
N <- 50
d <- sim(m,N); d$z <- rbinom(N,1,0.5)
dd <- mets::fast.reshape(d); dd$num <- dd$num+3
spaghetti(y~num,dd,id="id",lty=1,col=Col(1,.4),
          trend.formula=~factor(num),trend=TRUE,trend.col="darkblue")
dd$num <- dd$num+rnorm(nrow(dd),sd=0.5) ## Unbalance
spaghetti(y~num,dd,id="id",lty=1,col=Col(1,.4),
          trend=TRUE,trend.col="darkblue")
spaghetti(y~num,dd,id="id",lty=1,col=Col(1,.4),
           trend.formula=~num+I(num^2),trend=TRUE,trend.col="darkblue")
}
```

---

stack.estimate *Stack estimating equations*

Description

Stack estimating equations (two-stage estimator)

Usage

```
## S3 method for class 'estimate'
stack(
  x,
  model2,
  D1u,
  inv.D2u,
  propensity,
  dpropensity,
  U,
  keep1 = FALSE,
  propensity.arg,
  estimate.arg,
  na.action = na.pass,
  ...
)
```

Arguments

x	Model 1
model2	Model 2
D1u	Derivative of score of model 2 w.r.t. parameter vector of model 1
inv.D2u	Inverse of deri
propensity	propensity score (vector or function)
dpropensity	derivative of propensity score wrt parameters of model 1
U	Optional score function (model 2) as function of all parameters
keep1	If FALSE only parameters of model 2 is returned
propensity.arg	Arguments to propensity function
estimate.arg	Arguments to 'estimate'
na.action	Method for dealing with missing data in propensity score
...	Additional arguments to lower level functions

Value

Object of class `estimate` (see estimate.default) with inference adjusted for the two-stage estimation.

Examples

```
m <- lvm(z0~x)
Missing(m, z ~ z0) <- r~x
distribution(m,~x) <- binomial.lvm()
p <- c(r=-1,'r~x'=0.5,'z0~x'=2)
beta <- p[3]/2
d <- sim(m,500,p=p,seed=1)
m1 <- estimate(r~x,data=d,family=binomial)
```

```
d$w <- d$r/predict(m1,type="response")
m2 <- estimate(z~1, weights=w, data=d)
(e <- stack(m1,m2,propensity=TRUE))
```

`subset.lvm` *Extract subset of latent variable model*

Description

Extract measurement models or user-specified subset of model

Usage

```
## S3 method for class 'lvm'
subset(x, vars, ...)
```

Arguments

`x`	`lvm`-object.
`vars`	Character vector or formula specifying variables to include in subset.
`...`	Additional arguments to be passed to the low level functions

Value

A `lvm`-object.

Author(s)

Klaus K. Holst

Examples

```
m <- lvm(c(y1,y2)~x1+x2)
subset(m,~y1+x1)
```

summary.estimate *Summary of estimate objects*

Description

Computes hypothesis tests, contrasts, and small-sample corrections for an estimate object. The arguments `null`, `contrast`, `type`, and `var.adj` were previously available on `estimate.default()` and have been moved here.

Usage

```
## S3 method for class 'estimate'
summary(
  object,
  contrast,
  ...,
  null = 0,
  level = 0.95,
  type,
  var.adj = 0.25,
  df,
  transform = NULL,
  print = NULL
)
```

Arguments

`object`	an `estimate` object.
`contrast`	(optional) contrast matrix for the final Wald test.
`...`	additional arguments passed to contr.
`null`	(optional) null hypothesis to test.
`level`	level of confidence limits (default 0.95)
`type`	type of small-sample correction. Requires the estimate to have been computed with `IC=TRUE` (the default).
`var.adj`	variance adjustment parameter for small-sample correction. Requires the estimate to have been computed with `IC=TRUE` (the default).
`df`	degrees of freedom for t-based inference (default: `NULL` for Gaussian approximation; when set, confidence intervals and p-values use the t-distribution with `df` degrees of freedom)
`transform`	(optional) function applied to the point estimates and confidence interval bounds *after* inference is performed on the original scale. Useful for variance-stabilizing transformations, e.g., compute CIs on the `atanh` (Fisher z) scale and back-transform with `tanh`.
`print`	(optional) custom print function for the resulting `summary.estimate` object

See Also

estimate.default()

---

summary.sim *Summary method for 'sim' objects*

Description

Summary method for 'sim' objects

Usage

```
## S3 method for class 'sim'
summary(
  object,
  estimate = NULL,
  se = NULL,
  confint,
  true = NULL,
  fun,
  names = NULL,
  unique.names = TRUE,
  minimal = FALSE,
  level = 0.95,
  quantiles = c(0, 0.025, 0.5, 0.975, 1),
  df = Inf,
  ...
)
```

Arguments

object	sim object
estimate	(optional) columns with estimates
se	(optional) columns with standard error estimates
confint	(optional) list of pairs of columns with confidence limits
true	(optional) vector of true parameter values
fun	(optional) summary function
names	(optional) names of estimates
unique.names	if TRUE, unique.names will be applied to column names
minimal	if TRUE, minimal summary will be returned
level	confidence level (0.95)
quantiles	quantiles (0,0.025,0.5,0.975,1)
df	degrees of freedom in t-distribution used for constructing CIs (default Gaussian approximation)
...	additional levels to lower-level functions

timedep *Time-dependent parameters*

Description

Add time-varying covariate effects to model

Usage

```
timedep(object, formula, rate, timecut, type = "coxExponential.lvm", ...)
```

Arguments

object	Model
formula	Formula with rhs specifying time-varying covariates
rate	Optional rate parameters. If given as a vector this parameter is interpreted as the raw (baseline-)rates within each time interval defined by timecut. If given as a matrix the parameters are interpreted as log-rates (and log-rate-ratios for the time-varying covariates defined in the formula).
timecut	Time intervals
type	Type of model (default piecewise constant intensity)
...	Additional arguments to lower level functions

Author(s)

Klaus K. Holst

Examples

```
## Piecewise constant hazard
m <- lvm(y~1)
m <- timedep(m,y~1,timecut=c(0,5),rate=c(0.5,0.3))

## Not run:
d <- sim(m,1e4); d$status <- TRUE
dd <- mets::lifetable(Surv(y,status)~1,data=d,breaks=c(0,5,10));
exp(coef(glm(events ~ offset(log(atrisk)) + -1 + interval, dd, family=poisson)))

## End(Not run)


## Piecewise constant hazard and time-varying effect of z1
m <- lvm(y~1)
distribution(m,~z1) <- Binary.lvm(0.5)
R <- log(cbind(c(0.2,0.7,0.9),c(0.5,0.3,0.3)))
m <- timedep(m,y~z1,timecut=c(0,3,5),rate=R)
```

```
## Not run:
d <- sim(m,1e4); d$status <- TRUE
dd <- mets::lifetable(Surv(y,status)~z1,data=d,breaks=c(0,3,5,Inf));
exp(coef(glm(events ~ offset(log(atrisk)) + -1 + interval+z1:interval, dd, family=poisson)))

## End(Not run)



## Explicit simulation of time-varying effects
m <- lvm(y~1)
distribution(m,~z1) <- Binary.lvm(0.5)
distribution(m,~z2) <- binomial.lvm(p=0.5)
#variance(m,~m1+m2) <- 0
#regression(m,m1[m1:0] ~ z1) <- log(0.5)
#regression(m,m2[m2:0] ~ z1) <- log(0.3)
regression(m,m1 ~ z1,variance=0) <- log(0.5)
regression(m,m2 ~ z1,variance=0) <- log(0.3)
intercept(m,~m1+m2) <- c(-0.5,0)
m <- timedep(m,y~m1+m2,timecut=c(0,5))

## Not run:
d <- sim(m,1e5); d$status <- TRUE
dd <- mets::lifetable(Surv(y,status)~z1,data=d,breaks=c(0,5,Inf))
exp(coef(glm(events ~ offset(log(atrisk)) + -1 + interval + interval:z1, dd, family=poisson)))

## End(Not run)
```

`toformula` *Converts strings to formula*

Description

Converts a vector of predictors and a vector of responses (characters) i#nto a formula expression.

Usage

```
toformula(y = ".", x = ".")
```

Arguments

y	vector of predictors
x	vector of responses

Value

An object of class `formula`

Author(s)

Klaus K. Holst

See Also

as.formula,

Examples

```
toformula(c("age","gender"), "weight")
```

---

tr *Trace operator*

Description

Calculates the trace of a square matrix.

Usage

```
tr(x, ...)
```

Arguments

x	Square numeric matrix
...	Additional arguments to lower level functions

Value

numeric

Author(s)

Klaus K. Holst

See Also

crossprod, tcrossprod

Examples

```
tr(diag(1:5))
```

trim *Trim string of (leading/trailing/all) white spaces*

Description

Trim string of (leading/trailing/all) white spaces

Usage

```
trim(x, all = FALSE, ...)
```

Arguments

x	String
all	Trim all whitespaces?
...	additional arguments to lower level functions

Author(s)

Klaus K. Holst

twindata *Twin menarche data*

Description

Simulated data

id	numeric	Twin-pair id
zyg	character	Zygosity (MZ or DZ)
twinnum	numeric	Twin number (1 or 2)
agemena	numeric	Age at menarche (or censoring)
status	logical	Censoring status (observed:=T,censored:=F)
bw	numeric	Birth weight
msmoke	numeric	Did mother smoke? (yes:=1,no:=0)

Format

data.frame

Source

Simulated

twostage *Two-stage estimator*

Description

Generic function.

Usage

```
twostage(object, ...)
```

Arguments

object	Model object
...	Additional arguments to lower level functions

See Also

twostage.lvm twostage.lvmfit twostage.lvm.mixture twostage.estimate

twostage.lvmfit *Two-stage estimator (non-linear SEM)*

Description

Two-stage estimator for non-linear structural equation models

Usage

```
## S3 method for class 'lvmfit'
twostage(
  object,
  model2,
  data = NULL,
  predict.fun = NULL,
  id1 = NULL,
  id2 = NULL,
  all = FALSE,
  formula = NULL,
  std.err = TRUE,
  ...
)
```

Arguments

object	Stage 1 measurement model
model2	Stage 2 SEM
data	data.frame
predict.fun	Prediction of latent variable
id1	Optional id-variable (stage 1 model)
id2	Optional id-variable (stage 2 model)
all	If TRUE return additional output (naive estimates)
formula	optional formula specifying non-linear relation
std.err	If FALSE calculations of standard errors will be skipped
...	Additional arguments to lower level functions

Examples

```
m <- lvm(c(x1,x2,x3)~f1,f1~z,
         c(y1,y2,y3)~f2,f2~f1+z)
latent(m) <- ~f1+f2
d <- simulate(m,100,p=c("f2,f2"=2,"f1,f1"=0.5),seed=1)

## Full MLE
ee <- estimate(m,d)

## Manual two-stage
## Not run:
m1 <- lvm(c(x1,x2,x3)~f1,f1~z); latent(m1) <- ~f1
e1 <- estimate(m1,d)
pp1 <- predict(e1,f1~x1+x2+x3)

d$u1 <- pp1[,]
d$u2 <- pp1[,]^2+attr(pp1,"cond.var")[1]
m2 <- lvm(c(y1,y2,y3)~eta,c(y1,eta)~u1+u2+z); latent(m2) <- ~eta
e2 <- estimate(m2,d)

## End(Not run)

## Two-stage
m1 <- lvm(c(x1,x2,x3)~f1,f1~z); latent(m1) <- ~f1
m2 <- lvm(c(y1,y2,y3)~eta,c(y1,eta)~u1+u2+z); latent(m2) <- ~eta
pred <- function(mu,var,data,...)
    cbind("u1"=mu[,1],"u2"=mu[,1]^2+var[1])
(mm <- twostage(m1,model2=m2,data=d,predict.fun=pred))

if (interactive()) {
    pf <- function(p) p["eta"]+p["eta~u1"]*u + p["eta~u2"]*u^2
    plot(mm,f=pf,data=data.frame(u=seq(-2,2,length.out=100)),lwd=2)
}

 ## Reduce test timing
```

```
## Splines
f <- function(x) cos(2*x)+x+-0.25*x^2
m <- lvm(x1+x2+x3~eta1, y1+y2+y3~eta2, latent=~eta1+eta2)
functional(m, eta2~eta1) <- f
d <- sim(m,500,seed=1,latent=TRUE)
m1 <- lvm(x1+x2+x3~eta1,latent=~eta1)
m2 <- lvm(y1+y2+y3~eta2,latent=~eta2)
mm <- twostage(m1,m2,formula=eta2~eta1,type="spline")
if (interactive()) plot(mm)

nonlinear(m2,type="quadratic") <- eta2~eta1
a <- twostage(m1,m2,data=d)
if (interactive()) plot(a)

kn <- c(-1,0,1)
nonlinear(m2,type="spline",knots=kn) <- eta2~eta1
a <- twostage(m1,m2,data=d)
x <- seq(-3,3,by=0.1)
y <- predict(a, newdata=data.frame(eta1=x))

if (interactive()) {
  plot(eta2~eta1, data=d)
  lines(x,y, col="red", lwd=5)

  p <- estimate(a,f=function(p) predict(a,p=p,newdata=x))$coefmat
  plot(eta2~eta1, data=d)
  lines(x,p[,1], col="red", lwd=5)
  confband(x,lower=p[,3],upper=p[,4],center=p[,1], polygon=TRUE, col=Col(2,0.2))

  l1 <- lm(eta2~splines::ns(eta1,knots=kn),data=d)
  p1 <- predict(l1,newdata=data.frame(eta1=x),interval="confidence")
  lines(x,p1[,1],col="green",lwd=5)
  confband(x,lower=p1[,2],upper=p1[,3],center=p1[,1], polygon=TRUE, col=Col(3,0.2))
}
 ## Reduce test timing

## Not run:  ## Reduce timing
 ## Cross-validation example
 ma <- lvm(c(x1,x2,x3)~u,latent=~u)
 ms <- functional(ma, y~u, value=function(x) -.4*x^2)
 d <- sim(ms,500)#,seed=1)
 ea <- estimate(ma,d)

 mb <- lvm()
 mb1 <- nonlinear(mb,type="linear",y~u)
 mb2 <- nonlinear(mb,type="quadratic",y~u)
 mb3 <- nonlinear(mb,type="spline",knots=c(-3,-1,0,1,3),y~u)
 mb4 <- nonlinear(mb,type="spline",knots=c(-3,-2,-1,0,1,2,3),y~u)
 ff <- lapply(list(mb1,mb2,mb3,mb4),
      function(m) function(data,...) twostage(ma,m,data=data,st.derr=FALSE))
 a <- cv(ff,data=d,rep=1)
 a
```

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

`twostageCV` *Cross-validated two-stage estimator*

Description

Cross-validated two-stage estimator for non-linear SEM

Usage

```
twostageCV(
  model1,
  model2,
  data,
  control1 = list(trace = 0),
  control2 = list(trace = 0),
  knots.boundary,
  nmix = 1:4,
  df = 1:9,
  fix = TRUE,
  std.err = TRUE,
  nfolds = 5,
  rep = 1,
  messages = 0,
  ...
)
```

Arguments

`model1`	model 1 (exposure measurement error model)
`model2`	model 2
`data`	data.frame
`control1`	optimization parameters for model 1
`control2`	optimization parameters for model 1
`knots.boundary`	boundary points for natural cubic spline basis
`nmix`	number of mixture components
`df`	spline degrees of freedom
`fix`	automatically fix parameters for identification (TRUE)
`std.err`	calculation of standard errors (TRUE)
`nfolds`	Number of folds (cross-validation)
`rep`	Number of repeats of cross-validation
`messages`	print information (>0)
`...`	additional arguments to lower

Examples

```
 ## Reduce Ex.Timings##'
m1 <- lvm( x1+x2+x3 ~ u, latent= ~u)
m2 <- lvm( y ~ 1 )
m <- functional(merge(m1,m2), y ~ u, value=function(x) sin(x)+x)
distribution(m, ~u1) <- uniform.lvm(-6,6)
d <- sim(m,n=500,seed=1)
nonlinear(m2) <- y~u1
if (requireNamespace('mets', quietly=TRUE)) {
  set.seed(1)
  val <- twostageCV(m1, m2, data=d, std.err=FALSE, df=2:6, nmix=1:2,
                 nfolds=2)
  val
}
```

---

vars *Extract variable names from latent variable model*

Description

Extract exogenous variables (predictors), endogenous variables (outcomes), latent variables (random effects), manifest (observed) variables from a `lvm` object.

Usage

```
vars(x,...)

endogenous(x,...)

exogenous(x,...)

manifest(x,...)

latent(x,...)

## S3 replacement method for class 'lvm'
exogenous(x, xfree = TRUE,...) <- value

## S3 method for class 'lvm'
exogenous(x,variable,latent=FALSE,index=TRUE,...)

## S3 replacement method for class 'lvm'
latent(x,clear=FALSE,...) <- value
```

Arguments

x	lvm-object
...	Additional arguments to be passed to the low level functions
variable	list of variables to alter
latent	Logical defining whether latent variables without parents should be included in the result
index	For internal use only
clear	Logical indicating whether to add or remove latent variable status
xfree	For internal use only
value	Formula or character vector of variable names.

Details

vars returns all variables of the lvm-object including manifest and latent variables. Similarily manifest and latent returns the observered resp. latent variables of the model. exogenous returns all manifest variables without parents, e.g. covariates in the model, however the argument latent=TRUE can be used to also include latent variables without parents in the result. Pr. default lava will not include the parameters of the exogenous variables in the optimisation routine during estimation (likelihood of the remaining observered variables conditional on the covariates), however this behaviour can be altered via the assignment function exogenous<- telling lava which subset of (valid) variables to condition on. Finally latent returns a vector with the names of the latent variables in x. The assigment function latent<- can be used to change the latent status of variables in the model.

Value

Vector of variable names.

Author(s)

Klaus K. Holst

See Also

endogenous, manifest, latent, exogenous, vars

Examples

```
g <- lvm(eta1 ~ x1+x2)
regression(g) <- c(y1,y2,y3) ~ eta1
latent(g) <- ~eta1
endogenous(g)
exogenous(g)
identical(latent(g), setdiff(vars(g),manifest(g)))
```

var_ic *Variance based on influence function*

Description

Computes the empirical variance of the influence function, providing a sandwich-type variance estimator.

Usage

```
var_ic(x, ...)
```

Arguments

x	Influence function matrix (observations x parameters), typically obtained from IC.
...	Additional arguments (currently not used)

Value

Covariance matrix (p x p).

vec *vec operator*

Description

vec operator

Usage

```
vec(x, matrix = FALSE, sep = ".", ...)
```

Arguments

x	Array
matrix	If TRUE a row vector (matrix) is returned
sep	Seperator
...	Additional arguments

Details

Convert array into vector

Author(s)

Klaus Holst

wait *Wait for user input (keyboard or mouse)*

Description

Wait for user input (keyboard or mouse)

Usage

```
wait()
```

Author(s)

Klaus K. Holst

wkm *Weighted K-means*

Description

Weighted K-means via Lloyd's algorithm

Usage

```
wkm(
  x,
  mu,
  data,
  weights = rep(1, NROW(x)),
  iter.max = 20,
  n.start = 5,
  init = "kmpp",
  ...
)
```

Arguments

x	Data (or formula)
mu	Initial centers (or number centers chosen randomly among x)
data	optional data frmae
weights	Optional weights
iter.max	Max number of iterations
n.start	Number of restarts
init	method to create initial centres (default kmeans++)
...	Additional arguments to lower level functions

Author(s)

Klaus K. Holst

Examples

```
## Two well-separated Gaussian blobs in 2-D
set.seed(1)
x <- rbind(matrix(rnorm(100, mean = -3), ncol = 2),
           matrix(rnorm(100, mean =  3), ncol = 2))
res <- wkm(x, mu = 2)
table(res$cluster)
res$center

## Supply explicit initial centers (as a list)
res2 <- wkm(x, mu = list(c(-3, -3), c(3, 3)))

## Weighted clustering: up-weight the second blob
w <- c(rep(1, 50), rep(10, 50))
res3 <- wkm(x, mu = 2, weights = w)

## Formula interface on a data.frame
wkm(~ Sepal.Length + Sepal.Width, data = iris, mu = 3)
```

---

wrapvec *Wrap vector*

Description

Wrap vector

Usage

```
wrapvec(x, delta = 0L, ...)
```

Arguments

x	Vector or integer
delta	Shift
...	Additional parameters

Examples

```
wrapvec(5,2)
```

zibreg *Regression model for binomial data with unkown group of immortals*

Description

Regression model for binomial data with unkown group of immortals (zero-inflated binomial regression)

Usage

```
zibreg(
  formula,
  formula.p = ~1,
  data,
  family = stats::binomial(),
  offset = NULL,
  start,
  var = "hessian",
  ...
)
```

Arguments

formula	Formula specifying
formula.p	Formula for model of disease prevalence
data	data frame
family	Distribution family (see the help page family)
offset	Optional offset
start	Optional starting values
var	Type of variance (robust, expected, hessian, outer)
...	Additional arguments to lower level functions

Author(s)

Klaus K. Holst

Examples

```
## Simulation
n <- 2e3
x <- runif(n,0,20)
age <- runif(n,10,30)
z0 <- rnorm(n,mean=-1+0.05*age)
z <- cut(z0,breaks=c(-Inf,-1,0,1,Inf))
p0 <- lava:::expit(model.matrix(~z+age) %*% c(-.4, -.4, 0.2, 2, -0.05))
y <- (runif(n)<lava:::tigol(-1+0.25*x-0*age))*1
```

```
u <- runif(n)<p0
y[u==0] <- 0
d <- data.frame(y=y,x=x,u=u*1,z=z,age=age)
head(d)

## Estimation
e0 <- zibreg(y~x*z,~1+z+age,data=d)
e <- zibreg(y~x,~1+z+age,data=d)
compare(e,e0)
e
PD(e0,intercept=c(1,3),slope=c(2,6))

B <- rbind(c(1,0,0,0,20),
           c(1,1,0,0,20),
           c(1,0,1,0,20),
           c(1,0,0,1,20))
prev <- summary(e,pr.contrast=B)$prevalence

x <- seq(0,100,length.out=100)
newdata <- expand.grid(x=x,age=20,z=levels(d$z))
fit <- predict(e,newdata=newdata)
plot(0,0,type="n",xlim=c(0,101),ylim=c(0,1),xlab="x",ylab="Probability(Event)")
count <- 0
for (i in levels(newdata$z)) {
  count <- count+1
  lines(x,fit[which(newdata$z==i)],col="darkblue",lty=count)
}
abline(h=prev[3:4,1],lty=3:4,col="gray")
abline(h=prev[3:4,2],lty=3:4,col="lightgray")
abline(h=prev[3:4,3],lty=3:4,col="lightgray")
legend("topleft",levels(d$z),col="darkblue",lty=seq_len(length(levels(d$z))))
```

`%++%` *Concatenation operator*

Description

For matrices a block-diagonal matrix is created. For all other data types he operator is a wrapper of `paste`.

Usage

```
x %++% y
```

Arguments

`x`	First object
`y`	Second object of same class

Details

Concatenation operator

Author(s)

Klaus K. Holst

See Also

blockdiag, paste, cat,

Examples

```
## Block diagonal
matrix(rnorm(25),5)%++%matrix(rnorm(25),5)
## String concatenation
"Hello "%++%" World"
## Function composition
f <- log %++% exp
f(2)
```

%ni% *Matching operator (x not in y) oposed to the* %in%*-operator (x in y)*

Description

Matching operator

Usage

```
x %ni% y
```

Arguments

x	vector
y	vector of same type as x

Value

A logical vector.

Author(s)

Klaus K. Holst

See Also

match

Examples

```
1:10 %ni% c(1,5,10)
```

Index

∗ **algebra**
tr, 133
∗ **aplot**
labels<-, 72
∗ **color**
Col, 18
∗ **datagen**
sim.lvm, 119
∗ **datasets**
bmd, 8
bmidata, 9
brisa, 11
calcium, 14
deprdiag, 37
hubble, 61
hubble2, 62
indoorenv, 67
missingdata, 79
nldata, 88
nsem, 90
semdata, 116
serotonin, 116
twindata, 134
∗ **graphs**
Graph, 60
labels<-, 72
path, 93
∗ **hplot**
devcoords, 37
plot.lvm, 98
plotConf, 103
rotate2, 114
∗ **htest**
compare, 21
modelsearch, 83
∗ **iplot**
click, 15
confband, 23
pdfconvert, 96
wait, 142
∗ **manip**
NA2x, 87
∗ **math**
tr, 133
∗ **methods**
gof, 58
path, 93
∗ **misc**
%++%, 145
%ni%, 146
Grep, 60
∗ **models**
bootstrap.lvm, 10
complik, 22
confint.lvmfit, 26
constrain<-, 28
covariance, 33
estimate.lvm, 50
eventTime, 52
gof, 58
Graph, 60
intercept, 68
lava.options, 73
lvm, 74
mixture, 80
Model, 82
mvnmix, 85
partialcor, 92
path, 93
regression<-, 111
rmvar, 113
sim.lvm, 119
subset.lvm, 128
toformula, 132
vars, 139
∗ **regression**
bootstrap.lvm, 10
complik, 22

confint.lvmfit, 26
constrain<-, 28
covariance, 33
estimate.lvm, 50
eventTime, 52
intercept, 68
lvm, 74
mixture, 80
mvnmix, 85
partialcor, 92
plot.lvm, 98
plotConf, 103
regression<-, 111
rmvar, 113
sim.lvm, 119
subset.lvm, 128
vars, 139
∗ **survival**
eventTime, 52
∗ **utilities**
%++%, 145
%ni%, 146
Grep, 60
index, 65
makemissing, 75
rbind.Surv, 110
toformula, 132
%in.closed% *(%ni%)*, 146
%in.open% *(%ni%)*, 146
%++%, 145
%ni%, 146

aalenExponential.lvm *(sim.lvm)*, 119
addvar, 5
addvar<- *(addvar)*, 5
adjMat *(children)*, 15
agrep, *61*
alpha_zmax *(closed_testing)*, 17
ancestors *(children)*, 15
as.formula, *133*
as.sim *(sim.default)*, 117

backdoor, 5
baptize, 6
beta.lvm *(sim.lvm)*, 119
Binary.lvm *(sim.lvm)*, 119
binary.lvm *(sim.lvm)*, 119
binomial.lvm *(sim.lvm)*, 119
binomial.rd, 6
binomial.rr *(binomial.rd)*, 6
blockdiag, 8
bmd, 8
bmidata, 9
bootstrap, 9, *27*
bootstrap.lvm, 10
bootstrap.lvmfit *(bootstrap.lvm)*, 10
brisa, 11
By, 12

c.estimate, 12, *45*
c.estimate(), *78*
c.summary.estimate, 13
calcium, 14
cancel, 14
cancel<- *(cancel)*, 14
cat, *146*
categorical *(sim.lvm)*, 119
categorical<- *(sim.lvm)*, 119
children, 15
chisq.lvm *(sim.lvm)*, 119
click, 15
closed_testing, 17
Col, 18
colorbar, 19
colsel *(click)*, 15
Combine, 20
commutation, 21
compare, 21, *40*, *83*
complik, 22
confband, 23
confint.lvmfit, *11*, 26
confint.multigroupfit *(confint.lvmfit)*, 26
confpred, 27
constant.lvm *(sim.lvm)*, 119
constrain *(constrain<-)*, 28
constrain<-, 28
constraints *(constrain<-)*, 28
contr, 31, *45*, *129*
correlation, 32
covariance, *30*, 33, *75*
covariance<- *(covariance)*, 33
coxExponential.lvm *(sim.lvm)*, 119
coxGompertz.lvm *(sim.lvm)*, 119
coxWeibull.lvm *(sim.lvm)*, 119
crossprod, *133*
csplit, 35
curly, 36

density.sim (plot.sim), 100
deprdiag, 37
descendants (children), 15
devcoords, 37
diagtest, 38
Diff (diagtest), 38
distribution (sim.lvm), 119
distribution<- (sim.lvm), 119
dsep (dsep.lvm), 39
dsep.lvm, 39

edgelabels (labels<-), 72
edgelabels<- (labels<-), 72
edgeList (children), 15
effects (path), 93
endogenous, *140*
endogenous (vars), 139
equivalence, *22*, 40, *83*
estimate, *129*
estimate (estimate.default), 41
estimate.array, 41, *45*
estimate.data.frame (estimate.array), 41
estimate.default, *41*, 41, *48*, *49*, *78*, *127*
estimate.default(), *129*, *130*
estimate.formula, 48
estimate.glm, 49
estimate.list, 49
estimate.lvm, 50
eventTime, 52
eventTime<- (eventTime), 52
exogenous, *140*
exogenous (vars), 139
exogenous<- (vars), 139
Expand, 55

foldr (csplit), 35
forestplot (confband), 23
fplot, 56
functional (sim.lvm), 119
functional<- (sim.lvm), 119
future.apply::future_mapply(), *118*
future::plan(), *117*, *118*

Gamma.lvm (sim.lvm), 119
gamma.lvm (sim.lvm), 119
gaussian.lvm (sim.lvm), 119
getSAS, 57
gkgamma (multinomial), 84
GM2.lvm (sim.lvm), 119
GM3.lvm (sim.lvm), 119
gof, 58
Graph, 60, *82*
Graph<- (Graph), 60
Grep, 60
grep, *61*

heavytail (sim.lvm), 119
heavytail<- (sim.lvm), 119
hubble, 61
hubble2, 62

IC, *42*, *44*, *141*
IC (IC.default), 62
IC.default, 62
id.lvm (sim.lvm), 119
idplot, *16*
idplot (click), 15
iid, *62*, 63
images, 63
index, *65*, 65
index.lvm, 66
index.lvmfit (index.lvm), 66
index<-, *65*
index<- (index), 65
index<-.lvm (index.lvm), 66
index<-.lvmfit (index.lvm), 66
indoorenv, 67
information, *59*, 67
information.lvmfit (gof), 58
intercept, *30*, 68, *75*
intercept<- (intercept), 68
intervention (intervention.lvm), 69
intervention.lvm, 69
intervention<- (intervention.lvm), 69
Inverse, 70

kappa.multinomial (multinomial), 84
kappa.table (multinomial), 84
kill (rmvar), 113
kill<- (rmvar), 113
ksmooth2, 71

labels (labels<-), 72
labels<-, 72
latent, *140*
latent (vars), 139
latent<- (vars), 139
lava.options, 73

lm(), *104*
loggamma.lvm *(sim.lvm)*, 119
logit.lvm *(sim.lvm)*, 119
logLik.lvmfit *(gof)*, 58
lognormal.lvm *(sim.lvm)*, 119
lvm, 74

makemissing, 75
manifest, *140*
manifest *(vars)*, 139
match, *146*
measurement *(subset.lvm)*, 128
measurement.error, 76
merge.estimate, *13*, *45*, 77
merge.estimate(), *13*
Missing, 78
Missing, *(Missing)*, 78
Missing<- *(Missing)*, 78
missingdata, 79
mixture, 80
Model, *60*, 82
Model<- *(Model)*, 82
modelPar(), *66*
modelsearch, *22*, *40*, 83
moments *(gof)*, 58
multinomial, 84
multinomial.lvm *(sim.lvm)*, 119
mvn.lvm *(sim.lvm)*, 119
mvnmix, 85

na.fail(), *87*
na.omit(), *87*
na.pass(), *87*
na.pass0, 87
NA2x, 87
nldata, 88
nodecolor *(labels<-)*, 72
nodecolor<- *(labels<-)*, 72
none.lvm *(sim.lvm)*, 119
nonlinear *(twostage.lvmfit)*, 135
nonlinear<- *(twostage.lvmfit)*, 135
normal.lvm *(sim.lvm)*, 119
NR, 88
nsem, 90
numDeriv::jacobian, *44*

odds *(diagtest)*, 38
offdiag *(revdiag)*, 113
offdiag<- *(revdiag)*, 113
ones.lvm *(sim.lvm)*, 119
OR *(diagtest)*, 38
ordinal *(ordinal<-)*, 90
ordinal<-, 90
ordreg, 91

pairwise.diff, *45*
pairwise.diff *(contr)*, 31
parameter<- *(constrain<-)*, 28
parents *(children)*, 15
pareto.lvm *(sim.lvm)*, 119
parpos, 92
parsedesign, *45*
parsedesign *(contr)*, 31
partialcor, 92
paste, *146*
path, 93
pcor, 94
PD, 95
pdfconvert, 96
plot.estimate, 97
plot.lvm, 98
plot.lvmfit *(plot.lvm)*, 98
plot.sim, 100
plot.sim(), *118*
plot_region *(confband)*, 23
plotConf, 103
poisson.lvm *(sim.lvm)*, 119
predict.lvm, 105
predict.lvmfit *(predict.lvm)*, 105
predict_glm, 108
predictlvm, 107
Print, 109
print.lvm *(lvm)*, 74
probit.lvm *(sim.lvm)*, 119

Range.lvm, 109
Ratio *(diagtest)*, 38
rbind.Surv, 110
regression, *30*, *75*
regression *(regression<-)*, 111
regression<-, 111
revdiag, 113
revdiag<- *(revdiag)*, 113
riskcomp *(diagtest)*, 38
rmvar, 113
rmvar<- *(rmvar)*, 113
roots *(children)*, 15
rot2D *(rotate2)*, 114

rot3D *(rotate2)*, 114
rotate2, 114

scheffe, 115
score, *59*, 115
score.glm, *108*
score.lvmfit *(gof)*, 58
semdata, 116
Sequence.lvm *(sim.lvm)*, 119
serotonin, 116
sim *(sim.default)*, 117
sim.default, 117
sim.default(), *12, 13*
sim.lvm, 119
sim.lvm(), *118*
sim.lvmfit *(sim.lvm)*, 119
simulate.lvm *(sim.lvm)*, 119
simulate.lvmfit *(sim.lvm)*, 119
sinks *(children)*, 15
spaghetti, 124
stack.estimate, 126
stats::glm, *108*
stats::predict.glm, *108*
stats::vcov, *43*
student.lvm *(sim.lvm)*, 119
subset.lvm, 128
summary.estimate, *45*, 129
summary.lvm *(lvm)*, 74
summary.sim, 130
summary.sim(), *118*
surface *(ksmooth2)*, 71

tcrossprod, *133*
termplot(), *105*
test_wald *(compare)*, 21
test_wald(), *17*
threshold.lvm *(sim.lvm)*, 119
timedep, 131
timedep<- *(timedep)*, 131
toformula, 132
totaleffects *(path)*, 93
tr, 133
transform.lvm *(sim.lvm)*, 119
transform<- *(sim.lvm)*, 119
trim, 134
twindata, 134
twostage, 135
twostage.estimate *(twostage.lvmfit)*, 135
twostage.lvm *(twostage.lvmfit)*, 135
twostage.lvmfit, 135
twostageCV, 138

uniform.lvm *(sim.lvm)*, 119

var_ic, *62*, 141
variance *(covariance)*, 33
variance<- *(covariance)*, 33
vars, 139, *140*
vec, 141

wait, 142
waitclick *(wait)*, 142
weibull.lvm *(sim.lvm)*, 119
wkm, 142
wrapvec, 143

x2NA *(NA2x)*, 87

zibreg, 144