

Package ‘pscl’

November 14, 2006

Version 0.73

Type Package

Date 2006-11-14

Title Political Science Computational Laboratory, Stanford University

Author Simon Jackman <jackman@stanford.edu>, with contributions from Alex Tahk, Achim Zeileis, Christina Maimone and Jim Fearon

Maintainer Simon Jackman <jackman@stanford.edu>

Depends R (>= 2.3.0), MASS, stats, mvtnorm, coda

Suggests MCMCpack

Description Bayesian analysis of item-response theory (IRT) models, roll call analysis; computing highest density regions; maximum likelihood estimation of zero-inflated and hurdle models for count data; data sets used in writing and teaching at the Political Science Computational Laboratory; seats-votes curves

SaveImage yes

License GPL

URL <http://pscl.stanford.edu>

R topics documented:

AustralianElections	2
betaHPD	4
bioChemists	5
ca2006	6
computeMargins	7
constrain.items	8
constrain.legis	10
convertCodes	13
dropRollCall	14
dropUnanimous	15
extractRollCallObject	16
hurdle	17
hurdle.control	20

ideal	21
idealToMCMC	25
igamma	26
ntable	28
odTest	29
partycodes	31
plot.ideal	31
plot.predict.ideal	33
plot.seatsVotes	34
postProcess	36
predict.hurdle	38
predict.ideal	40
predict.zeroinfl	42
predprob	43
predprob.glm	44
predprob.ideal	45
prussian	46
readKH	47
rollcall	49
s109	52
sc9497	53
seatsVotes	54
state.info	55
summary.ideal	56
summary.rollcall	58
tracex	59
unionDensity	61
vectorRepresentation	62
vuong	63
zeroinfl	64
zeroinfl.control	67
Index	69

AustralianElections

elections to Australian House of Representatives, 1949-2004

Description

Aggregate data on the 23 elections to Australia's House of Representatives, 1949 to 2004.

Usage

```
data(AustralianElections)
```

Format

date date of election, stored using the `Date` class
Seats numeric, number of seats in the House of Representatives
Uncontested numeric, number of uncontested seats
ALPSeats numeric, number of seats won by the Australian Labor Party
LPSeats numeric, number of seats won by the Liberal Party
CPSeats numeric, number of seats won by the Country Party (later known as the National Party)
OtherSeats numeric, number of seats won by other parties and/or independent candidates
ALP numeric, percentage of first preference votes cast for Australian Labor Party candidates
ALP2PP numeric, percentage of the two-party preferred vote won by Australian Labor Party candidates
LP numeric, percent of first preference votes cast for Liberal Party candidates
NP numeric, percent of first preference votes cast for National Party (Country Party) candidates
DLP numeric, percent of first preference votes cast for Democratic Labor Party candidates
Dem numeric, percent of first preference votes cast for Australian Democrat candidates
Green numeric, percent of first preference votes cast for Green Party candidates
Hanson numeric, percent of first preference votes cast for candidates from Pauline Hanson's One Nation party
Com numeric, percent of first preference votes cast for Communist Party candidates
AP numeric, percent of first preference votes cast for Australia Party candidates
Informal numeric, percent of ballots cast that are spoiled, blank, or otherwise uncountable (usually because of errors in enumerating preferences)
Turnout numeric, percent of enrolled voters recorded as having turned out to vote (Australia has compulsory voting)

Source

Australian Electoral Commission. <http://www.aec.gov.au>.

Examples

```

data(AustralianElections)
attach(AustralianElections)
alpSeatShare <- ALPSeats/Seats
alpVoteShare <- ALP2PP/100

## log-odds transforms
x <- log(alpVoteShare/(1-alpVoteShare))
y <- log(alpSeatShare/(1-alpSeatShare))

ols <- lm(y~x) ## Tufte-style seats-votes regression

xseq <- seq(-4.5,4.5,length=500)
yhat <- coef(ols)[1] + coef(ols)[2]*xseq
yhat <- exp(yhat)/(1+exp(yhat))
xseq <- exp(xseq)/(1+exp(xseq))

## seats vote curve
plot(x=alpVoteShare,

```

```

y=alpSeatShare,
xlab="ALP Vote Share",
ylab="ALP Seat Share")
lines(xseq,yhat,lwd=2)
abline(h=.5,lty=2)
abline(v=.5,lty=2)

```

betaHPD

compute and optionally plot beta HDRs

Description

Compute and optionally plot highest density regions for the Beta distribution.

Usage

```
betaHPD(alpha,beta,p=.95,plot=FALSE,xlim=NULL,debug=FALSE)
```

Arguments

alpha	scalar, first shape parameter of the Beta density. Must be greater than 1, see details
beta	scalar, second shape parameter of the Beta density. Must be greater than 1, see details
p	scalar, content of HPD, must lie between 0 and 1
plot	logical flag, if TRUE then plot the density and show the HDR
xlim	numeric vector of length 2, the limits of the density's support to show when plotting; the default is NULL, in which case the function will confine plotting to where the density is non-negligible
debug	logical flag, if TRUE produce messages to the console

Details

The Beta density arises frequently in Bayesian models of binary events, rates, and proportions, which take on values in the open unit interval. For instance, the Beta density is a conjugate prior for the unknown success probability in binomial trials. With shape parameters $\alpha > 1$ and $\beta > 1$, the Beta density is unimodal.

In general, suppose $\theta \in \Theta \subseteq R^k$ is a random variable with density $f(\theta)$. A highest density region (HDR) of $f(\theta)$ with content $p \in (0, 1]$ is a set $Q \subseteq \Theta$ with the following properties:

$$\int_Q f(\theta) d\theta = p$$

and

$$f(\theta) > f(\theta^*) \forall \theta \in Q, \theta^* \notin Q.$$

For a unimodal Beta density (the class of Beta densities handled by this function), a HDR of content $0 < p < 1$ is simply an interval $Q \in (0, 1)$.

This function uses numerical methods to solve for the end points of a HDR for a Beta density with user-specified shape parameters, via repeated calls to the functions `dbeta`, `pbeta` and `qbeta`. The function `optimize` is used to find points v and w such that

$$f(v) = f(w)$$

subject to the constraint

$$\int_v^w f(\theta; \alpha, \beta) d\theta = p,$$

where $f(\theta; \alpha, \beta)$ is a Beta density with shape parameters α and β .

In the special case of $\alpha = \beta > 1$, the end points of a HDR with content p are given by the $(1 \pm p)/2$ quantiles of the Beta density, and are computed with the `qbeta` function.

Again note that the function will only compute a HDR for a unimodal Beta density, and exit with an error if `alpha <= 1 | beta <= 1`. Note that the uniform density results with $\alpha = \beta = 1$, which does not have a unique HDR with content $0 < p < 1$. With shape parameters $\alpha < 1$ and $\beta > 1$ (or vice-versa, respectively), the Beta density is infinite at 0 (or 1, respectively), but still integrates to one, and so a HDR is still well-defined (but not implemented here, at least not yet). Similarly, with $0 < \alpha, \beta < 1$ the Beta density is infinite at both 0 and 1, but integrates to one, and again a HDR of content $p < 1$ is well-defined in this case, but will be a set of two disjoint intervals (again, at present, this function does not cover this case).

Value

If the numerical optimization is successful an vector of length 2, containing v and w , defined above. If the optimization fails for whatever reason, a vector of `NA`s is returned.

The function will also produce a plot of the density with area under the density supported by the HDR shaded, if the user calls the function with `plot=TRUE`; the plot will appear on the current graphics device.

Debugging messages are printed to the console if the `debug` logical flag is set to `TRUE`.

Author(s)

Simon Jackman (`jackman@stanford.edu`). Thanks to John Bullock who discovered a bug in an earlier version.

See Also

`pbeta`, `qbeta`, `dbeta`, `uniroot`

Examples

```
betaHPD(4, 5)
betaHPD(2, 120)
betaHPD(120, 45, p=.75, xlim=c(0, 1))
```

bioChemists	<i>article production by graduate students in biochemistry Ph.D. programs</i>
-------------	---

Description

A sample of 915 biochemistry graduate students.

Usage

```
data(bioChemists)
```

Format

art count of articles produced during last 3 years of Ph.D.
 fem factor indicating gender of student, with levels Men and Women
 mar factor indicating marital status of student, with levels Single and Married
 kid5 number of children aged 5 or younger
 phd prestige of Ph.D. department
 ment count of articles produced by Ph.D. mentor during last 3 years

Source

found in Stata format at <http://www.indiana.edu/~jslsoc/stata/socdata/couart2.dta>

References

Long, J. Scott (1990). The origins of sex difference in science. *Social Forces*. 68:1297-1315.
 Long, J. Scott (1997). *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, California: Sage.

ca2006	<i>California Congressional Districts in 2006</i>
--------	---

Description

Election returns and identifying information, California's 53 congressional districts in the 2006 Congressional elections.

Usage

```
data(ca2006)
```

Format

A data frame with 53 observations on the following 11 variables.

district numeric, number of Congressional district
D numeric, number of votes for the Democratic candidate
R numeric, votes for the Republican candidate
Other numeric, votes for other candidates
IncParty character, party of the incumbent (or retiring member), D or R
IncName character, last name of the incumbent, character NA if no incumbent running
open logical, TRUE if no incumbent running
contested logical, TRUE if both major parties ran candidates
Bush2004 numeric, votes for George W. Bush (R) in the district in the 2004 presidential election
Kerry2004 numeric, votes for John Kerry (D) in 2004
Other2004 numeric votes for other candidates in 2004
Bush2000 numeric, votes for George W. Bush in 2000
Gore2000 numeric, votes for Al Gore (D) in 2000

Source

2006 data from the California Secretary of State's web site, <http://vote.ss.ca.gov>Returns/usrep/all.htm> (last updated Tuesday November 14, 2006). 2004 and 2000 presidential vote in congressional districts from the *2006 Almanac of American Politics*.

References

Michael Baraon and Richard E. Cohen. 2006. *The Almanac of American Politics, 2006*. National Journal Group: Washington, D.C.

Examples

```
data(ca2006)

## 2006 CA congressional vote against 2004 pvote
y <- ca2006$D/(ca2006$D+ca2006$R)
x <- ca2006$Kerry2004/(ca2006$Kerry2004+ca2006$Bush2004)

pch <- rep(19,length(y))
pch[ca2006$open] <- 1
col <- rep("black",length(y))
col[11] <- "red"      ## Pembo (R) loses to McNerney (D)
plot(y~x,pch=pch,
     col=col,
     xlim=range(x,y,na.rm=TRUE),
     ylim=range(x,y,na.rm=TRUE),
     xlab="Kerry Two-Party Vote, 2004",
     ylab="Democratic Two-Party Vote Share, 2006")
abline(0,1)
abline(h=.5,lty=2)
abline(v=.5,lty=2)
legend(x="topleft",
      bty="n",
```

```
col=c("red","black","black"),
pch=c(19,19,1),
legend=c("Seat Changing Hands",
         "Seat Retained by Incumbent Party",
         "Open Seat (no incumbent running)")
)
```

computeMargins *add information about voting outcomes to a rollcall object*

Description

Add summaries of each roll call vote to a `rollcall` object.

Usage

```
computeMargins(object, dropList = NULL)
```

Arguments

<code>object</code>	an object of class <code>rollcall</code>
<code>dropList</code>	a <code>list</code> (or <code>alist</code>) listing voting decisions, legislators and/or votes to be dropped from the analysis; see <code>dropRollCall</code> for details.

Details

The subsetting implied by the `dropList` is first applied to the `rollcall` object, via `dropRollCall`. Then, for each remaining roll call vote, the number of legislators voting “Yea”, “Nay”, and not voting are computed, using the encoding information in the `codes` component of the `rollcall` object via the `convertCodes` function. The matrix of vote counts are added to the `rollcall` object as a component `voteMargins`.

Value

An object of class `rollcall`, with a component `voteMargins` that is a matrix with four columns:

Yea	number of legislators voting “Yea”
Nay	number of legislators voting “Nay”
NA	number of legislators not voting “Nay”
Min	the number of legislators voting on the losing side of the roll call

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

`dropRollCall` on specifying a `dropList`. The vote-specific marginals produced by this function are used by as `dropRollCall`, `summary.ideal` and `predict.ideal`.

Examples

```

data(s109)
tmp <- computeMargins(s109)
dim(tmp$voteMargins)  ## 556 by 4

tmp <- computeMargins(s109,
                      dropList=list(codes="notInLegis",lop=0))
dim(tmp$voteMargins)  ## 477 by 4

```

constrain.items *constrain item parameters in analysis of roll call data*

Description

Sets constraints on specified item parameters in Bayesian analysis of roll call data by generating appropriate priors and start values for Markov chain Monte Carlo iterations.

Usage

```

constrain.items(obj, dropList = list(codes = "notInLegis", lop = 0),
               x, d = 1)

```

Arguments

obj	an object of class <code>rollcall</code> .
dropList	a <code>list</code> (or <code>alist</code>) indicating which voting decisions, legislators and/or roll calls are to be excluded from the subsequent analysis; see <code>dropRollCall</code> for details.
x	a <code>list</code> containing elements with names matching votes found in <code>dimnames(object\$votes)[[1]]</code> (but after any subsetting specified by <code>dropList</code>). Each component of the list must be a vector containing <code>d</code> elements, specifying the value to which the item discrimination parameters should be constrained, in each of the <code>d</code> dimensions. The intercept or item difficulty parameter will not be constrained.
d	numeric, positive integer, the number of dimensions for which to set up the priors and start values.

Details

`constrain.items` and its cousin, `constrain.legis` are usefully thought of as “pre-processor” functions, generating priors *and* start values for both the item parameters and the ideal points. For the items specified in `x`, the prior mean for each dimension is set to the value given in `x`, and the prior precision for each dimension is set to `1e12` (i.e., a near-degenerate “spike” prior). For the other items, the priors are set to a mean of 0 and precision 0.01. All of the ideal points are given normal priors with mean 0, precision 1.

Start values are also generated for both ideal points and item parameters. The start values for the items specified in `x` are set to the values specified in `x`. The list resulting from `constrain.items` can then be given as the value for the parameters `priors` and `startvals` when `ideal` is run. The user is responsible for ensuring that a sufficient number of items are constrained such that when `ideal` is run, the model parameters are identified.

`dropRollCall` is first called to generate the desired roll call matrix. The entries of the roll call matrix are mapped to `c(0, 1, NA)` using the `codes` component of the `rollcall` object. See the discussion in the documentation of `ideal` for details on the generation of start values.

Value

a list with elements:

<code>xp</code>	prior means for ideal points. A matrix of dimensions number of legislators in <code>obj</code> by <code>d</code> .
<code>xpv</code>	prior meansprecisions for ideal points. A matrix of dimensions number of legislators in <code>obj</code> by <code>d</code> .
<code>bp</code>	prior means for item parameters. A matrix of dimensions number of items or votes in <code>obj</code> by <code>d+1</code> .
<code>bpv</code>	prior meansprecisions for item parameters. A matrix of dimensions number of items or votes in <code>obj</code> by <code>d+1</code> .
<code>xstart</code>	start values for ideal points. A matrix of dimensions number of legislators in <code>obj</code> by <code>d</code> .
<code>bstart</code>	start values for ideal points. A matrix of dimensions number of items or votes in <code>obj</code> by <code>d+1</code> .

See Also

[rollcall](#), [ideal](#), [constrain.legis](#)

Examples

```
## run ideal with the default parameters
data(s109)
## Not run:
## run a 1d model, look for lack of fit
idl <- ideal(s109,
            d=1,
            meanzero=TRUE,
            store.item=TRUE,
            maxiter=1e5,
            burnin=1e3,
            thin=1e2)
idlsum <- summary(idl, include.beta=TRUE)

suspect1 <- idlsum$bSig[[1]]=="95
close60 <- idlsum$bResults[[1]][,"Yea"] < 60
close40 <- idlsum$bResults[[1]][,"Yea"] > 40
suspect <- suspect1 & close60 & close40
idlsum$bResults[[1]][suspect,]
suspectVotes <- dimnames(idlsum$bResults[[1]][suspect,])[1]
## End(Not run)

## constraints on 2d model,
## close rollcall poorly fit by 1d model
## serves as reference item for 2nd dimension

cl <- constrain.items(s109,
                    x=list("2-150"=c(0,7),
```

```

      "2-169"=c(7,0)),
      d=2)

## Not run:
idlConstrained <- ideal(s109,
                        d=2,
                        meanzero=TRUE,
                        priors=c1,
                        startvals=c1,
                        maxiter=1e5,
                        burnin=1e3,
                        thin=1e2)
summary(idlConstrained,include.beta=TRUE)
## End(Not run)

```

constrain.legis *constrain legislators' ideal points in analysis of roll call data*

Description

Sets constraints on specified legislators for ideal point estimation by generating appropriate priors and start values.

Usage

```
constrain.legis(obj, dropList = list(codes = "notInLegis", lop = 0),
               x, d = 1)
```

Arguments

obj	an object of class <code>rollcall</code> .
dropList	a <code>list</code> (or <code>alist</code>) indicating which voting decisions, legislators and/or roll calls are to be excluded from the subsequent analysis; see <code>dropRollCall</code> for details.
x	a <code>list</code> containing elements with names matching legislators found in <code>dimnames(object\$votes)</code> (but after any subsetting specified by <code>dropList</code>). Each element must be a vector containing <code>d</code> elements, specifying the value to which the ideal point should be constrained in each of <code>d</code> dimensions. <code>x</code> must have at least <code>d+1</code> components; i.e., supplying a necessary (but not sufficient) set of constraints for global identification of the parameters of a <code>d</code> -dimensional item-response model, see <code>Details</code> .
d	the number of dimensions for which to set up the priors and start values.

Details

`constrain.items` and its cousin, `constrain.legis` are usefully thought of as “pre-processor” functions, generating priors *and* start values for both the item parameters and the ideal points.

For the legislators specified in `x`, the prior mean for each dimension is set to the specified value and the prior precision for each dimension is set to $1e12$ (i.e., a near-degenerate “spike” prior). For the other legislators, the priors on their ideal points are set to a mean of 0 and precision .01. All of the item parameter priors are set to mean 0, precision 0.01.

Start values are also generated for both ideal points and item parameters. The start values for the legislators specified in `x` are set to the values specified in `x`. The list resulting from `constrain.legis` can then be given as the value for the parameters `priors` and `startvals` when `ideal` is run. `constrain.legis` requires that $d+1$ constraints be specified; if the constrained ideal points are linearly independent, then the parameters of the item-response model are (at least locally) identified. For instance, when fitting a 1 dimensional model, constraining the ideal points of two legislators is sufficient to globally identify the model parameters.

`dropRollCall` is first called to generate the desired roll call matrix. The entries of the roll call matrix are mapped to `c(0, 1, NA)` using the `codes` component of the `rollcall` object. See the discussion in the documentation of `ideal` for details on the generation of start values.

Value

a list with elements:

<code>xp</code>	prior means for ideal points. A matrix of dimensions number of legislators in <code>rc</code> by <code>d</code> .
<code>xpv</code>	prior meansprecisions for ideal points. A matrix of dimensions number of legislators in <code>rc</code> by <code>d</code> .
<code>bp</code>	prior means for item parameters. A matrix of dimensions number of items or votes in <code>rc</code> by <code>d+1</code> .
<code>bpv</code>	prior meansprecisions for item parameters. A matrix of dimensions number of items or votes in <code>rc</code> by <code>d+1</code> .
<code>xstart</code>	start values for ideal points. A matrix of dimensions number of legislators in <code>rc</code> by <code>d</code> .
<code>bstart</code>	start values for ideal points. A matrix of dimensions number of items or votes in <code>rc</code> by <code>d+1</code> .

See Also

[rollcall](#), [ideal](#), [constrain.items](#)

Examples

```
data(s109)
cl <- constrain.legis(s109,
  x=list("KENNEDY (D MA)"=-1,
        "ENZI (R WY)"=1),
  d=1)

## short run for examples
idlConstrained <- ideal(s109,
  d=1,
  priors=cl,      ## use cl
  startvals=cl,  ## use cl
  maxiter=500,
  burnin=100,
  thin=10)
summary(idlConstrained)

cl2 <- constrain.legis(s109,
  x=list("KENNEDY (D MA)"=c(-1,0),
        "ENZI (R WY)"=c(1,0),
```

```

      "CHAFEE (R RI)"=c(0,-.5)),
d=2)

## Not run:
## too long for examples
id2Constrained <- ideal(s109,
  d=2,
  priors=c12,      ## priors (w constraints)
  startvals=c12,  ## start value (w constraints)
  store.item=TRUE,
  maxiter=5000,
  burnin=500,
  thin=25)

## End(Not run)
## short run for examples
id2Constrained <- ideal(s109,
  d=2,
  priors=c12,      ## priors (w constraints)
  startvals=c12,  ## start value (w constraints)
  store.item=TRUE,
  maxiter=500,
  burnin=100,
  thin=10)
summary(id2Constrained,include.items=TRUE)

```

 convertCodes

convert entries in a rollcall matrix to binary form

Description

Convert roll call matrix to binary form using encoding information.

Usage

```
convertCodes(object, codes = object$codes)
```

Arguments

object	rollcall object
codes	list, mapping entries in the votes component of rollcall object to 0 ('Nay'), 1 ('Yea') and NA (missing, abstentions, etc). Defaults to the codes component of the rollcall object.

Details

See [rollcall](#) for details on the form of the codes list.

Value

a [matrix](#) with dimensions equal to the dimensions of the votes component of the [rollcall](#) object.

Note

Any entries in the `votes` matrix that can not be mapped into `c(0, 1, NA)` using the information in `codes` are mapped to `NA`, with an informative message sent to the console.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[rollcall](#)

Examples

```
data(s109)
mat <- convertCodes(s109)
table(mat, exclude=NULL)
```

dropRollCall	<i>drop user-specified elements from a rollcall object</i>
--------------	--

Description

Drop user-specified elements of rollcall object, returning a roll call object.

Usage

```
dropRollCall(object, dropList)
```

Arguments

<code>object</code>	an object of class <code>rollcall</code>
<code>dropList</code>	a <code>list</code> (or <code>alist</code>) with some (or all) of the following components:
<code>codes</code>	character or numeric, possibly a vector. If character, it should match the names of <code>object\$codes</code> , indicating the set of entries in <code>object\$votes</code> to be set to <code>NA</code> . If numeric, then <code>codes</code> indicates the entries in <code>object\$votes</code> that will be set to <code>NA</code> .
<code>lop</code>	numeric, non-negative integer, less than number of legislators represented in <code>object</code> . Roll calls with <code>lop</code> or fewer legislators voting in the minority are dropped.
<code>legisMin</code>	numeric, non-negative integer, less than number of roll calls represented in <code>object</code> . Legislators with <code>legisMin</code> or fewer votes are dropped.
<code>dropLegis</code>	an <code>expression</code> that evaluates to mode <code>logical</code> , vector of length equal to the number of legislators represented in <code>object</code> . The expression is evaluated in the <code>legis.data</code> component of the rollcall object. Legislators for whom the expression evaluates to <code>TRUE</code> are dropped.
<code>dropVotes</code>	an <code>expression</code> that evaluates to mode <code>logical</code> , vector of length equal to the number of rollcalls represented in <code>object</code> . The expression is evaluated in the <code>votes.data</code> component of the rollcall object. Rollcalls for which the expression evaluates to <code>TRUE</code> are dropped.

Details

It is often desirable to restrict the analysis of roll call data in various ways. For one thing, unanimous votes provide no information discriminating among legislators: hence, a useful default for summary and analysis is `dropList=list(lop=0)`. See the examples for other possibilities, limited only by the information supplied in `legis.data` and `votes.data`.

Value

An object of class `rollcall` with components modified by the subsetting indicated in the `dropList`.

Note

With the exception of `codes`, each component of `dropList` generates a vector of mode `logical`, either with respect to legislators or votes. These logical vectors are then combined element-wise, such that if any one of the subsetting restrictions is `TRUE` for a particular legislator or vote, then that legislator or vote is dropped. Some summaries are reported to the console along the way.

When `dropList` uses the `dropLegis` or `dropVotes` components then `dropList` should be constructed as an `alist` command; this ensures that the `dropLegis` and `dropVotes` components of `dropList` are objects of mode `expression`, and `evaluated` to mode `logical` in the `legis.data` and `vote.data` `environments` by the function, if possible (rather than being evaluated immediately in the environment calling `dropRollCall` or constructing `dropList`). See the examples. This is not entirely satisfactory, and behavior more like the `subset` argument in function `lm` would be preferable.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[dropUnanimous](#), [summary.rollcall](#), [ideal](#), [alist](#).

Examples

```
data(s109)
s109.working <- dropRollCall(s109,
                           dropList=list(lop=0))
s109.working <- dropRollCall(s109,
                           dropList=list(lop=0,
                                           code="notInLegis"))
s109.working <- dropRollCall(s109,
                           dropList=list(lop=3,
                                           code="notInLegis"))

## note use of alist, since dropLegis is an expression
dropList <- alist(lop=3,
                 dropLegis=party!="D",
                 code="notInLegis")
s109.working <- dropRollCall(s109,dropList=dropList)

## Not run:
## read 102nd House from Poole web site
h102 <- readKH("ftp://voteview.ucsd.edu/dtaord/hou102kh.ord")
```

```
## drop President from roll call matrix
h102 <- dropRollCall(h102,
                    dropList=alist(dropLegis=state=="USA"))
## End(Not run)
```

dropUnanimous *drop unanimous votes from rollcall objects and matrices*

Description

Drop unanimous votes from rollcall objects and rollcall matrices.

Usage

```
dropUnanimous(obj, lop = 0)
```

Arguments

obj	object, either of class <code>rollcall</code> or <code>matrix</code>
lop	numeric, non-negative integer, less than number of legislators represented in obj. Roll calls with lop or fewer legislators voting in the minority are dropped. Default is 0, meaning that unanimous votes are dropped.

Details

Unanimous votes are the equivalent of test items that all subjects score “correct” (or all subjects scores “incorrect”); since there is no variation among the legislators/subjects, these votes/items provide no information as to latent traits (ideology, preferences, ability). A reasonably large number of rollcalls in any contemporary U.S. Congress are unanimous.

Specific methods are provided for objects of class `rollcall` or `matrix`.

Value

A `rollcall` object or a link{`matrix`} depending on the class of object.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

`dropRollCall`, `rollcall`, `summary.rollcall`, `ideal`

Examples

```
data(s109)
s109.working <- dropUnanimous(s109)
summary(s109.working)
```

```
extractRollCallObject
```

return the roll call object used in fitting an ideal model

Description

Given a fitted model of class `ideal`, return the `rollcall` object that was used in the model fitting (i.e., apply all subsetting and recoding implied by the `dropList` passed to `ideal`).

Usage

```
extractRollCallObject(object)
```

Arguments

`object` an object of class `ideal`

Details

This function is used by many post-estimation commands that operate on objects of class `ideal`. The function inspects the `call` attribute of the `ideal` object, extracting the name of the `rollcall` object and the `dropList`, then hands them over to `dropRollCall`.

Value

An object of class `rollcall`

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

`rollcall`; see `dropRollCall` for details on the form of a `dropList`.

Examples

```
data(s109)
idl <- ideal(s109,
            d=1,
            meanzero=TRUE,
            maxiter=500,      ## short run for demo purposes
            burnin=100,
            thin=10)

tmp <- extractRollCallObject(idl)
summary(tmp)
v <- convertCodes(tmp)      ## roll call matrix per se
```

hurdle

*Hurdle Models for Count Data Regression***Description**

Fit hurdle regression models for count data via maximum likelihood.

Usage

```
hurdle(formula, data, subset, na.action,
        dist = c("poisson", "negbin", "geometric"),
        zero.dist = c("binomial", "poisson", "negbin", "geometric"),
        link = c("logit", "probit", "cloglog", "cauchit", "log"),
        control = hurdle.control(...),
        model = TRUE, y = TRUE, x = FALSE, ...)
```

Arguments

formula	symbolic description of the model, see details.
data, subset, na.action	arguments controlling formula processing via model.frame .
dist	character specification of count model family.
zero.dist	character specification of the zero hurdle model family.
link	character specification of link function in the binomial zero hurdle (only used if zero.dist = "binomial").
control	a list of control arguments specified via hurdle.control .
model, y, x	logicals. If TRUE the corresponding components of the fit (model frame, response, model matrix) are returned.
...	arguments passed to hurdle.control in the default setup.

Details

Hurdle count models are two-component models with a truncated count component for positive counts and a hurdle component that models the zero counts. Thus, unlike zero-inflation models, there are *not* two sources of zeros: the count model is only employed if the hurdle for modeling the occurrence of zeros is exceeded. The count model is typically a truncated Poisson or negative binomial regression (with log link). The geometric distribution is a special case of the negative binomial with size parameter equal to 1. For modeling the hurdle (occurrence of positive counts) either a binomial model can be employed or a censored count distribution. Binomial logit and censored geometric models as the hurdle part both lead to the same likelihood function and thus to the same coefficient estimates.

The formula can be used to specify both components of the model: If a formula of type $y \sim x_1 + x_2$ is supplied, then the same regressors are employed in both components. This is equivalent to $y \sim x_1 + x_2 \mid x_1 + x_2$. Of course, a different set of regressors could be specified for the zero hurdle component, e.g., $y \sim x_1 + x_2 \mid z_1 + z_2 + z_3$ giving the count data model $y \sim x_1 + x_2$ conditional on (|) the zero hurdle model $y \sim z_1 + z_2 + z_3$.

All parameters are estimated by maximum likelihood using [optim](#), with control options set in [hurdle.control](#). Starting values can be supplied, otherwise they are estimated by [glm.fit](#)

(the default). By default, the two components of the model are estimated separately using two `optim` calls. Standard errors are derived numerically using the Hessian matrix returned by `optim`. See `hurdle.control` for details.

The returned fitted model object is of class `"hurdle"` and is similar to fitted `"glm"` objects. For elements such as `"coefficients"` or `"terms"` a list is returned with elements for the zero and count components, respectively. For details see below.

A set of standard extractor functions for fitted model objects is available for objects of class `"hurdle"`, including methods to the generic functions `print`, `summary`, `coef`, `vcov`, `logLik`, `residuals`, `predict`, `fitted`, `terms`, `model.matrix`. See `predict.hurdle` for more details on all methods.

Value

An object of class `"hurdle"`, i.e., a list with components including

<code>coefficients</code>	a list with elements <code>"count"</code> and <code>"zero"</code> containing the coefficients from the respective models,
<code>residuals</code>	a vector of raw residuals (observed - fitted),
<code>fitted.values</code>	a vector of fitted means,
<code>optim</code>	a list (of lists) with the output(s) from the <code>optim</code> call(s) for minimizing the negative log-likelihood(s),
<code>control</code>	the control arguments passed to the <code>optim</code> call,
<code>start</code>	the starting values for the parameters passed to the <code>optim</code> call(s),
<code>n</code>	number of observations,
<code>df.null</code>	residual degrees of freedom for the null model ($= n - 2$),
<code>df.residual</code>	residual degrees of freedom for fitted model,
<code>terms</code>	a list with elements <code>"count"</code> , <code>"zero"</code> and <code>"full"</code> containing the terms objects for the respective models,
<code>theta</code>	estimate of the additional θ parameter of the negative binomial model(s) (if negative binomial component is used),
<code>SE.logtheta</code>	standard error(s) for $\log(\theta)$,
<code>loglik</code>	log-likelihood of the fitted model,
<code>vcov</code>	covariance matrix of all coefficients in the model (derived from the Hessian of the <code>optim</code> output(s)),
<code>dist</code>	a list with elements <code>"count"</code> and <code>"zero"</code> with character strings describing the respective distributions used,
<code>link</code>	character string describing the link if a binomial zero hurdle model is used,
<code>linkinv</code>	the inverse link function corresponding to <code>link</code> ,
<code>converged</code>	logical indicating successful convergence of <code>optim</code> ,
<code>call</code>	the original function call,
<code>formula</code>	the original formula,
<code>levels</code>	levels of the categorical regressors,
<code>contrasts</code>	a list with elements <code>"count"</code> and <code>"zero"</code> containing the contrasts corresponding to <code>levels</code> from the respective models,
<code>model</code>	the full model frame (if <code>model = TRUE</code>),
<code>y</code>	the response count vector (if <code>y = TRUE</code>),
<code>x</code>	a list with elements <code>"count"</code> and <code>"zero"</code> containing the model matrices from the respective models (if <code>x = TRUE</code>).

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

- Cameron, A. Colin and Pravin K. Trivedi. 1998. *Regression Analysis of Count Data*. New York: Cambridge University Press.
- Cameron, A. Colin and Pravin K. Trivedi 2005. *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.
- Mullahy, J. 1986. Specification and Testing of Some Modified Count Data Models. *Journal of Econometrics*. V33: 341–365.

See Also

[hurdle.control](#), [glm](#), [glm.fit](#), [glm.nb](#), [zeroinfl](#)

Examples

```
## from Long (1997)
data("bioChemists", package = "pscl")

## logit-poisson
## "art ~ ." is the same as "art ~ . | .", i.e.
## "art ~ fem + mar + kid5 + phd + ment | fem + mar + kid5 + phd + ment"
fm_hp1 <- hurdle(art ~ ., data = bioChemists)
summary(fm_hp1)

## geometric-poisson
fm_hp2 <- hurdle(art ~ ., data = bioChemists, zero = "geometric")
summary(fm_hp2)

## logit and geometric model are equivalent
coef(fm_hp1, model = "zero") - coef(fm_hp2, model = "zero")

## logit-negbin
fm_hnb1 <- hurdle(art ~ ., data = bioChemists, dist = "negbin")
summary(fm_hnb1)

## negbin-negbin
fm_hnb2 <- hurdle(art ~ ., data = bioChemists, dist = "negbin", zero = "negbin")
summary(fm_hnb2)
```

hurdle.control

Control Parameters for Hurdle Count Data Regression

Description

Various parameters that control fitting of hurdle regression models using [hurdle](#).

Usage

```
hurdle.control(method = "BFGS", maxit = 10000, trace = FALSE,
              separate = TRUE, start = NULL, ...)
```

Arguments

method	characters string specifying the method argument passed to <code>optim</code> .
maxit	integer specifying the maxit argument (maximal number of iterations) passed to <code>optim</code> .
trace	logical or integer controlling whether tracing information on the progress of the optimization should be produced (passed to <code>optim</code>).
separate	logical. Should the estimation of the parameters in the truncated count component and hurdle zero component be carried out separately? See details.
start	an optional list with elements "count" and "zero" (and potentially "theta") containing the coefficients for the corresponding component.
...	arguments passed to <code>optim</code> .

Details

All parameters in `hurdle` are estimated by maximum likelihood using `optim` with control options set in `hurdle.control`. Most arguments are passed on directly to `optim`, only `trace` is also used within `hurdle` and `separate/start` control how `optim` is called.

Starting values can be supplied via `start` or estimated by `glm.fit` (default). If `separate = TRUE` (default) the likelihoods of the truncated count component and the hurdle zero component will be maximized separately, otherwise the joint likelihood is set up and maximized. Standard errors are derived numerically using the Hessian matrix returned by `optim`. To supply starting values, `start` should be a list with elements "count" and "zero" and potentially "theta" (a named vector, for models with negative binomial components only) containing the starting values for the coefficients of the corresponding component of the model.

Value

A list with the arguments specified.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

See Also

[hurdle](#)

Examples

```
data("bioChemists", package = "pscl")

## default start values
fm1 <- hurdle(art ~ fem + ment, data = bioChemists, dist = "negbin", zero = "negbin")

## user-supplied start values
fm2 <- hurdle(art ~ fem + ment, data = bioChemists, dist = "negbin", zero = "negbin",
  start = list(count = c(0.3, -0.2, 0), zero = c(7, -2, 0.8), theta = c(count = 2, zero =
```

ideal	<i>analysis of roll call data (IRT models) via Markov chain Monte Carlo methods</i>
-------	---

Description

Analysis of `rollcall` data via the spatial voting model; analogous to fitting educational testing data via an item-response model. Model fitting via Markov chain Monte Carlo (MCMC).

Usage

```
ideal(object, codes = object$codes,
      dropList = list(codes = "notInLegis", lop = 0),
      d = 1, maxiter = 10000, thin = 100, burnin = 5000,
      impute = FALSE, meanzero = FALSE,
      priors = NULL, startvals = NULL,
      store.item = FALSE, file = NULL)
```

Arguments

object	an object of class <code>rollcall</code>
codes	a <code>list</code> describing the types of voting decisions in the roll call matrix (the votes component of the <code>rollcall</code> object); defaults to <code>object\$codes</code> , the codes in the <code>rollcall</code> object.
dropList	a <code>list</code> (or <code>alist</code>) listing voting decisions, legislators and/or votes to be dropped from the analysis; see <code>dropRollCall</code> for details.
d	numeric, (small) positive integer (defaults to 1).
maxiter	numeric, positive integer, multiple of <code>thin</code>
thin	numeric, positive integer, thinning interval used for recording MCMC iterations.
burnin	number of MCMC iterations to run before recording. The iteration numbered <code>burnin</code> will be recorded. Must be a multiple of <code>thin</code> .
impute	<code>logical</code> , whether to treat missing entries of the rollcall matrix as missing at random, sampling from the predictive density of the missing entries at each MCMC iteration.
meanzero	<code>logical</code> , whether estimated ideal points should have a mean of zero and standard deviation one. If <code>TRUE</code> , any user-supplied priors will be ignored. This option is helpful for unidimensional models, and is sufficient to locally identify the model parameters in this case; more restrictions are required for identification when $d > 1$. See Details.
priors	a <code>list</code> of parameters (means and variances) specifying normal priors for the legislators' ideal points. The default is <code>NULL</code> , in which case prior values will be generated for both legislators' ideal points and roll call parameters (for the ideal points, the default prior parameters are mean zero and variance one; for the item parameters the defaults are mean zero and variance 100). If not <code>NULL</code> , <code>priors</code> must be a <code>list</code> containing the elements <code>xp</code> , <code>xpv</code> , <code>bp</code> , <code>bpv</code> , which should be matrices: where
xp	a n by d matrix of prior <i>means</i> for the legislators' ideal points
xpv	a n by d matrix of prior <i>precisions</i> (inverse variances)

<code>bp</code>	a m by $d+1$ matrix of prior means for the item parameters (with the item difficulty parameter coming last)
<code>bpv</code>	a m by $d+1$ matrix of prior precisions for the item parameters. None of these elements may contain NA.
<code>startvals</code>	a list containing start values for legislators' ideal points and item parameters. Default is NULL, in which case start values will be generated for both legislators' ideal points and item parameters. See Details. If not NULL, <code>startvals</code> must be a list containing the elements <code>xstart</code> and <code>bstart</code> , which should be matrices. <code>xstart</code> must be of dimensions equal to the number of individuals (legislators) by d . <code>bstart</code> must be of dimensions number of items (votes) by $d+1$. <code>xstart</code> and <code>bstart</code> cannot contain NA.
<code>store.item</code>	logical, whether item discrimination parameters should be stored. Storing item discrimination parameters can consume a large amount of memory.
<code>file</code>	string, file to write MCMC output. Default is NULL, in which case MCMC output is stored in memory. Note that post-estimation commands like <code>plot</code> will not work unless MCMC output is stored in memory.

Details

The function fits a $d+1$ parameter item-response model to the roll call data object, so in one dimension the model reduces to the two-parameter item-response model popular in educational testing. See References.

Identification: The model parameters are **not identified** without the user supplying some restrictions on the model parameters (translations, rotations and re-scalings of the ideal points are observationally equivalent, via offsetting transformations of the item parameters). It is the user's responsibility to impose these restrictions; the following brief discussion provides some guidance.

For one-dimensional models, a simple route to identification is the `meanzero` option, which guarantees *local* identification (identification up to a 180 rotation of the recovered dimension). Near-degenerate "spike" priors (priors with arbitrarily large precisions) or the `constrain.legis` option on any two legislators' ideal points ensures *global* identification.

Identification in higher dimensions can be obtained by supplying fixed values for $d+1$ legislators' ideal points, provided the supplied points span a d -dimensional space (e.g., three supplied ideal points form a triangle in $d=2$ dimensions), via the `constrain.legis` option. In this case the function defaults to vague normal priors, but at each iteration the sampled ideal points are transformed back into the space of identified parameters, applying the linear transformation that maps the $d+1$ fixed ideal points from their sampled values to their fixed values.

Alternatively, one can impose restrictions on the item parameters via `constrain.items`.

Another route to identification is via *post-processing*. That is, the user can run `ideal` without any identification constraints, but then use the function `postProcess` to map the MCMC output from the space of unidentified parameters into the subspace of identified parameters.

Start values. Start values can be supplied by the user, or generated by the function itself. `constrain.legis` or `constrain.items` generate start values using the procedures discussed below, but also impose any (identifying) constraints imposed by the user. Start values for legislators' ideal points are generated by double-centering the roll call matrix (subtracting row means, and column means, adding in the grand mean), forming a correlation matrix across legislators, and extracting the first d eigenvectors, scaled by the square root of the corresponding eigenvalues. Any constraints from `constrain.legis` are then considered, with the unconstrained start values (linearly) transformed via least squares regression, minimizing the sum of the squared differences between the constrained and the unconstrained start values.

To generate start values for the rollcall/item parameters, a series of `binomial glms` are estimated (with a probit `link`), one for each rollcall/item, $j = 1, \dots, m$. The votes on the j -th rollcall/item are binary responses (presumed to be conditionally independent given each legislator's latent preference), and the (constrained or unconstrained) start values for legislators are used as predictors. The estimated coefficients from these probit models are stored to serve as start values for the item discrimination and difficulty parameters. Any constraints on particular item discrimination parameters from `constrain.legis` are then imposed.

Value

a `list` of class `ideal` with named components

<code>n</code>	<code>numeric</code> , integer, number of legislators in the analysis, after any subsetting via processing the <code>dropList</code> .
<code>m</code>	<code>numeric</code> , integer, number of rollcalls in roll call matrix, after any subsetting via processing the <code>dropList</code> .
<code>d</code>	<code>numeric</code> , integer, number of dimensions fitted.
<code>x</code>	a <code>matrix</code> containing the MCMC samples for the ideal point of each legislator in each dimension for each iteration from <code>burnin</code> to <code>maxiter</code> , at an interval of <code>thin</code> . Rows of the <code>x</code> matrix index iterations; columns index legislators.
<code>beta</code>	a <code>matrix</code> containing the MCMC samples for the item discrimination parameter for each item in each dimension, plus an intercept, for each iteration from <code>burnin</code> to <code>maxiter</code> , at an interval of <code>thin</code> . Rows of the <code>beta</code> matrix index MCMC iterations; columns index parameters.
<code>xbar</code>	a <code>matrix</code> containing the means of the MCMC samples for the ideal point of each legislator in each dimension, using iterations <code>burnin</code> to <code>maxiter</code> , at an interval of <code>thin</code> ; i.e., the column means of <code>x</code> .
<code>betabar</code>	a <code>matrix</code> containing the means of the MCMC samples for the vote-specific parameters, using iterations <code>burnin</code> to <code>maxiter</code> , at an interval of <code>thin</code> ; i.e., the column means of <code>beta</code> .
<code>call</code>	an object of class <code>call</code> , containing the arguments passed to <code>ideal</code> as unevaluated expressions.

Author(s)

Simon Jackman (jackman@stanford.edu), with help from Christina Maimone and Alex Tahk.

References

- Albert, James. 1992. Bayesian Estimation of normal ogive item response curves using Gibbs sampling. *Journal of Educational Statistics*. 17:251-269.
- Clinton, Joshua, Simon Jackman and Douglas Rivers. 2004. The Statistical Analysis of Roll Call Data. *American Political Science Review*. 98:335-370.
- Patz, Richard J. and Brian W. Junker. 1999. A Straightforward Approach to Markov Chain Monte Carlo Methods for Item Response Models. *Journal of Education and Behavioral Statistics*. 24:146-178.
- Rivers, Douglas. 2003. "Identification of Multidimensional Item-Response Models." Typescript. Department of Political Science, Stanford University.

See Also

[rollcall](#), [summary.ideal](#), [plot.ideal](#), [predict.ideal](#). [tracex](#) for graphical display of MCMC iterative history.

[idealToMCMC](#) converts the MCMC iterates in an `ideal` object to a form that can be used by the coda library.

[constrain.items](#) and [constrain.legis](#) for implementing identifying restrictions.

[postProcess](#) for imposing identifying restrictions *ex post*.

[MCMCirtId](#) and [MCMCirtKd](#) in the `MCMCpack` package provide similar functionality to `ideal`.

Examples

```
data(s109)

## short run for examples
idl <- ideal(s109,
            d=1,
            meanzero=TRUE,
            store.item=TRUE,
            maxiter=500,
            burnin=100,
            thin=10)

summary(idl)

## Not run:
## more realistic long run
idLong <- ideal(s109,
               d=1,
               meanzero=TRUE,
               store.item=TRUE,
               maxiter=251e3,
               burnin=1000,
               thin=1e3)

## End(Not run)
```

`idealToMCMC`

convert an object of class `ideal` to a coda MCMC object

Description

Converts the `x` element of an `ideal` object to an MCMC object, as used in the coda package.

Usage

```
idealToMCMC(object, burnin=NULL)
```

Arguments

`object` an object of class `ideal`.

`burnin` of the recorded MCMC samples, how many to discard as burnin? Default is `NULL`, in which case the value of `burnin` in the `ideal` object is used.

Value

A `mcmc` object as used by the `coda` package, starting at iteration `start`, drawn from the `x` component of the `ideal` object.

Note

When specifying a value of `burnin` different from that used in fitting the `ideal` object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the n -th iteration stored in an `ideal` object will not be iteration n if the user specified `thin>1` in the call to `ideal`. Here, iterations are tagged with their iteration number. Thus, if the user called `ideal` with `thin=10` and `burnin=100` then the stored iterations are numbered 100, 110, 120, Any future subsetting via a `burnin` refers to this iteration number.

See Also

`ideal`, `mcmc`

Examples

```
data(s109)
idl <- ideal(s109,
            d=1,
            meanzero=TRUE,
            maxiter=500, ## short run for demo purposes
            burnin=100,
            thin=10)
idl coda <- idealToMCMC(idl)
summary(idl coda)
```

igamma

inverse-Gamma distribution

Description

Density, distribution function, quantile function, and highest density region calculation for the inverse-Gamma distribution with parameters `alpha` and `beta`.

Usage

```
densigamma(x, alpha, beta)
pigamma(q, alpha, beta)
qigamma(p, alpha, beta)
rigamma(n, alpha, beta)
igammaHDR(alpha, beta, content=.95, debug=FALSE)
```

Arguments

<code>x, q</code>	vector of quantiles
<code>p</code>	vector of probabilities
<code>n</code>	number of random samples in <code>rigamma</code>
<code>alpha, beta</code>	rate and shape parameters of the inverse-Gamma density, both positive
<code>content</code>	scalar, $0 < \text{content} < 1$, volume of highest density region
<code>debug</code>	logical; if TRUE, debugging information from the search for the HDR is printed

Details

The inverse-Gamma density arises frequently in Bayesian analysis of normal data, as the (marginal) conjugate prior for the unknown variance parameter. The inverse-Gamma density for $x > 0$ with parameters $\alpha > 0$ and $\beta > 0$ is

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} \exp(-\beta/x)$$

where $\Gamma(x)$ is the [gamma](#) function

$$\Gamma(a) = \int_0^\infty t^{a-1} \exp(-t) dt$$

and so ensures $f(x)$ integrates to one. The inverse-Gamma density has a mean at $\beta/(\alpha - 1)$ for $\alpha > 1$ and has variance $\beta^2/((\alpha - 1)^2(\alpha - 2))$ for $\alpha > 2$. The inverse-Gamma density has a unique mode at $\beta/(\alpha + 1)$.

The evaluation of the density, cumulative distribution function and quantiles is done by calls to the `dgamma`, `pgamma` and `igamma` functions, with the arguments appropriately transformed. That is, note that if $x \sim IG(\alpha, \beta)$ then $1/x \sim G(\alpha, \beta)$.

Highest Density Regions. In general, suppose x has a density $f(x)$, where $x \in \Theta$. Then a highest density region (HDR) for x with content $p \in (0, 1]$ is a region (or set of regions) $\mathcal{Q} \subseteq \Theta$ such that:

$$\int_{\mathcal{Q}} f(x) dx = p$$

and

$$f(x) > f(x^*) \forall x \in \mathcal{Q}, x^* \notin \mathcal{Q}.$$

For a continuous, unimodal density defined with respect to a single parameter (like the inverse-Gamma case considered here with parameters $0 < \alpha < \infty$, $0 < \beta < \infty$), a HDR region Q of content p (with $0 < p < 1$) is a unique, closed interval on the real half-line.

This function uses numerical methods to solve for the boundaries of a HDR with content p for the inverse-Gamma density, via repeated calls the functions `densigamma`, `pigamma` and `qigamma`. In particular, the function `uniroot` is used to find points v and w such that

$$f(v) = f(w)$$

subject to the constraint

$$\int_v^w f(x; \alpha, \beta) d\theta = p.$$

Value

`densigamma` gives the density, `pigamma` the distribution function, `qigamma` the quantile function, `rigamma` generates random samples, and `igammaHDR` gives the lower and upper limits of the HDR, as defined above (NAs if the optimization is not successful).

Note

The `densigamma` is named so as not to conflict with the `digamma` function in the R [base](#) package (the derivative of the gamma function).

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[gamma](#), [dgamma](#), [pgamma](#), [qgamma](#), [uniroot](#)

Examples

```

alpha <- 4
beta <- 30
summary(rigamma(n=1000,alpha,beta))

xseq <- seq(.1,30,by=.1)
fx <- densigamma(xseq,alpha,beta)
plot(xseq,fx,type="n",
     xlab="x",
     ylab="f(x)",
     ylim=c(0,1.01*max(fx)),
     yaxs="i",
     axes=FALSE)
axis(1)
title(substitute(list(alpha==a,beta==b),list(a=alpha,b=beta)))
q <- igammaHDR(alpha,beta,debug=TRUE)
xlo <- which.min(abs(q[1]-xseq))
xup <- which.min(abs(q[2]-xseq))
plotZero <- par()$usr[3]
polygon(x=xseq[c(xlo,xlo:xup,xup:xlo)],
       y=c(plotZero,
          fx[xlo:xup],
          rep(plotZero,length(xlo:xup))),
       border=FALSE,
       col=gray(.45))
lines(xseq,fx,lwd=1.25)

## Not run:
alpha <- beta <- .1
xseq <- exp(seq(-7,30,length=1001))
fx <- densigamma(xseq,alpha,beta)
plot(xseq,fx,
     log="xy",
     type="l",
     ylim=c(min(fx),1.01*max(fx)),
     yaxs="i",
     xlab="x, log scale",
     ylab="f(x), log scale",
     axes=FALSE)
axis(1)

title(substitute(list(alpha==a,beta==b),list(a=alpha,b=beta)))
q <- igammaHDR(alpha,beta,debug=TRUE)
xlo <- which.min(abs(q[1]-xseq))
xup <- which.min(abs(q[2]-xseq))
plotZero <- min(fx)
polygon(x=xseq[c(xlo,xlo:xup,xup:xlo)],
       y=c(plotZero,
          fx[xlo:xup],
          rep(plotZero,length(xlo:xup))),
       border=FALSE,
       col=gray(.45))

```

```
lines(xseq,fx,lwd=1.25)
## End(Not run)
```

ntable	<i>nicey formatted tables</i>
--------	-------------------------------

Description

Nicely formatted tables, with row or column marginals etc.

Usage

```
ntable(x,y=NULL,
       percent=1,digits=2,
       row=FALSE,col=FALSE)
```

Arguments

x	vector or factor
y	vector of factor
percent	integer, 1 for row percentages (default), 2 for column percentages
digits	integer, digits to print after decimal place (default is 2)
row	logical, if TRUE, print row marginals
col	logical, if TRUE, print column marginals

Details

A wrapper function to [prop.table](#) that produces prettier looking results.

Value

nothing returned; the function prints the table and exits silently.

Author(s)

Jim Fearon (jfearon@stanford.edu)

See Also

[prop.table](#), [table](#)

Examples

```
data(bioChemists)
attach(bioChemists)
ntable(fem)
ntable(fem,mar,row=TRUE)
ntable(fem,mar,per=2,col=TRUE)
ntable(fem,mar,per=2,row=TRUE,col=TRUE)
```

`odTest`*likelihood ratio test for over-dispersion in count data*

Description

Compares the log-likelihoods of a negative binomial regression model and a Poisson regression model.

Usage

```
odTest(glmobj, digits = max(3, getOption("digits") - 3))
```

Arguments

<code>glmobj</code>	an object of class <code>negbin</code> produced by <code>glm.nb</code>
<code>digits</code>	number of digits in printed output

Details

The negative binomial model relaxes the assumption in the Poisson model that the (conditional) variance equals the (conditional) mean, by estimating one extra parameter. A likelihood ratio (LR) test can be used to test the null hypothesis that the restriction implicit in the Poisson model is true. The LR test-statistic has a non-standard distribution, even asymptotically, since the negative binomial over-dispersion parameter (called `theta` in `glm.nb`) is restricted to be positive. The asymptotic distribution of the LR test-statistic has probability mass of one half at zero, and a half χ_1^2 distribution above zero. This means that if testing at the $p = .05$ level, one should not reject the null unless the LR test statistic exceeds the critical value associated with the $p = .025$ level; this LR test involves just one parameter restriction, so the critical value of the test statistic at the $p = .05$ level is 5.02, instead of the usual 3.8 (i.e., the .975 quantile of the χ_1^2 distribution, versus the .95 quantile). This function simply reports the tail area (probability mass) to the right of the test statistic under the χ_1^2 distribution.

A Poisson model is run using `glm` with family set to `link{poisson}`, using the `formula` in the `negbin` model object passed as input. The `logLik` functions are used to extract the log-likelihood for each model.

Value

None; prints results and returns silently

Author(s)

Simon Jackman (jackman@stanford.edu)

References

A. Colin Cameron and Pravin K. Trivedi (1998) *Regression analysis of count data*. New York: Cambridge University Press.

See Also

[glm.nb](#), [logLik](#)

Examples

```
data(bioChemists)
require(MASS)
modelnb <- glm.nb(art ~ .,
                  data=bioChemists,
                  trace=TRUE)
odTest(modelnb)
```

partycodes	<i>political parties appearing in the U.S. Congress</i>
------------	---

Description

Numeric codes and names of 85 political parties appearing in Poole and Rosenthal's collection of U.S. Congressional roll calls.

Usage

```
data(partycodes)
```

Format

`code` integer, numeric code for legislator appearing in Poole and Rosenthal rollcall data files
`party` character, name of party

Details

The function [readKH](#) converts the integer codes into strings, via a table lookup in this data frame.

Source

Keith Poole's website: <http://voteview.com/party3.dat>

See Also

[readKH](#)

plot.ideal	<i>plots an ideal object</i>
------------	------------------------------

Description

Plot of the results of an ideal point estimation contained in an object of class `ideal`.

Usage

```
## S3 method for class 'ideal':
plot(x, conf.int=0.95, burnin=NULL, ...)

plot1d(x, d=1, conf.int=0.95, burnin=NULL,
       showAllNames = FALSE, ...)

plot2d(x, d1=1, d2=2, burnin=NULL,
       overlayCuttingPlanes=FALSE, ...)
```

Arguments

<code>x</code>	an object of class <code>ideal</code>
<code>conf.int</code>	for "ideal" objects with 1 dimension estimated, the level of the confidence interval to plot around the posterior mean for each legislator. If 2 or more dimensions were estimated, <code>conf.int</code> is ignored.
<code>d</code>	integer, which dimension to display in a 1d plot, if the object is a multidimensional ideal object
<code>burnin</code>	of the recorded MCMC samples, how many to discard as burnin? Default is NULL, in which case the value of <code>burnin</code> in the <code>ideal</code> object is used.
<code>showAllNames</code>	logical, if TRUE, the vertical axis will the names of all legislators. Default is FALSE to reduce clutter on typical-sized graph.
<code>d1</code>	integer, the number of the first dimension to plot when plotting multi-dimensional <code>ideal</code> objects. This dimension will appear on the horizontal (x) axis.
<code>d2</code>	integer, the number of the second dimension to plot when plotting multi-dimensional <code>ideal</code> objects. This dimension will appear on the vertical (y) axis.
<code>overlayCuttingPlanes</code>	logical, if TRUE, overlay the estimated bill-specific cutting planes
<code>...</code>	other parameters to be passed through to plotting functions.

Details

If the `ideal` object comes from fitting a $d=1$ dimensional model, then `plot.ideal` plots the mean of the posterior density over each legislator's ideal point, accompanied by a `conf.int` confidence interval. In this case, `plot.ideal` is simply a wrapper function to `plot1d`.

If the `ideal` object has $d=2$ dimensions, then `plot2d` is called, which plots the (estimated) mean of the posterior density of each legislator's ideal point (i.e., the ideal point/latent trait is a point in 2-dimensional Euclidean space, and the posterior density for each ideal point is a bivariate density). Single dimension summaries of the estimated ideal points (latent traits) can be obtained for multidimensional `ideal` objects by passing the `ideal` object directly to `plot1d` with `d` set appropriately.

If the `ideal` object has $d>2$ dimensions, a scatterplot matrix is produced via `pairs`, with the posterior means of the ideal points (latent traits) plotted against one another, dimension by dimension.

For unidimensional and two-dimensional models, if party information is available in the `rollcall` object contained in the `ideal` object, legislators from different parties are plotted in different colors. If the `ideal` object has more than 2 dimensions, `plot.ideal()` produces a matrix of plots of the mean ideal points of each dimension against the posterior mean ideal points of the other dimensions.

Note

When specifying a value of `burnin` different from that used in fitting the `ideal` object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the n -th iteration stored in an `ideal` object will not be iteration n if the user specified `thin>1` in the call to `ideal`. Here, iterations are tagged with their iteration number. Thus, if the user called `ideal` with `thin=10` and `burnin=100` then the stored iterations are numbered 100, 110, 120, Any future subsetting via a `burnin` refers to this iteration number.

See Also

`ideal`; `tracex` for trace plots, a graphical aid useful in diagnosing convergence of the MCMC algorithms.

Examples

```
data(s109)
id1 <- ideal(s109,
            d=1,
            meanzero=TRUE,
            store.item=TRUE,
            maxiter=500,  ## short run for examples
            burnin=100,
            thin=10)

plot(id1)

## Not run:
id2 <- ideal(s109,
            d=2,
            store.item=TRUE,
            maxiter=1e4,
            burnin=0,
            thin=25)

plot(id2, overlayCuttingPlanes=TRUE)

id2pp <- postProcess(id2,
                    constraints=list(BOXER=c(-1,0),
                                    INHOFE=c(1,0),
                                    CHAFEE=c(0,.25)))

plot(id2, overlayCuttingPlanes=TRUE)
## End(Not run)
```

`plot.predict.ideal` *plot methods for predictions from ideal objects*

Description

Plot classification success rates by legislators, or by roll calls, using predictions from `ideal`.

Usage

```
plot.predict.ideal(x, type = c("legis", "votes"),...)
```

Arguments

`x` an object of class `predict.ideal`.
`type` string; one of `legis` or `votes`.
`...` further arguments passed to or from other methods.

Details

`type="legis"` produces a plot of the “percent correctly predicted” for each legislator/subject (using the classification threshold set in `predict.ideal`) against the estimated ideal point of each legislator/subject (the estimated mean of the posterior density of the ideal point), dimension at a time. If the legislators’ party affiliations are available in the `rollcall` object that was passed to `ideal`, then legislators from the same party are plotted with a unique color.

`type="votes"` produces a plot of classification rates for each roll call, by the percentage of legislators voting for the losing side. The x-ordinate is jittered for clarity.

Value

After drawing plots on the current device, exits silently returning `invisible(NULL)`.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[predict.ideal ideal](#)

Examples

```
data(s109)
idl <- ideal(s109,
            d=1,
            meanzero=TRUE,      ## local identification in ld
            store.item=TRUE,    ## need this for predictions
            maxiter=500,        ## short run for demo purposes
            burnin=100,
            thin=10)
phat <- predict(idl)
plot(phat,type="legis")
plot(phat,type="votes")
```

`plot.seatsVotes` *plot seats-votes curves*

Description

Plot seats-votes curves produced by `seatsVotes`

Usage

```
plot.seatsVotes(x, type = c("seatsVotes", "density"),
               legend = "bottomright", transform=FALSE, ...)
```

Arguments

<code>x</code>	an object of class <code>seatsVotes</code>
<code>type</code>	character, partially matching the options above; see details
<code>legend</code>	where to put the legend when plotting with <code>type="seatsVotes"</code>
<code>transform</code>	logical, whether to transform the vote shares for <code>type="density"</code> ; see Details
<code>...</code>	arguments passed to or from other functions (e.g., options for the <code>density</code> function, when <code>type="density"</code>)

Details

A seats-votes curve (with various annotations) is produced with option `type="seatsVotes"`.

A density plot of the vote shares is produced with `type="density"`. A bimodal density corresponds to an electoral system with a proliferation of safe seats for both parties, and a seats-votes curve that is relatively flat (or “unresponsive”) in the neighborhood of average district-level vote shares of 50 percent. The density is fitted using the defaults in the `density` function, but with the density constrained to fall to zero at the extremes of the data, via the `from` and `to` options to `density`. A `rug` is added to the density plot.

If `transform=TRUE`, the vote shares are transformed prior to plotting, so as to reduce the extent to which extreme vote shares close to zero or 1 determine the shape of the density (i.e., this option is available only for plots of `type="density"`). The transformation is a sinusoidal function, a scaled “log-odds/inverse-log-odds” function mapping from (0,1) to (0,1): i.e., $f(x) = g(k \cdot h(x))$ where $h(x)$ is the log-odds transformation $h(x) = \log(x/(1-x))$, k is a scaling parameter set to $\sqrt{3}$, and $g(x)$ is the inverse-log-odds transformation $g(x) = \exp(x)/(1 + \exp(x))$. Note that this transformation is cosmetic, with the effect of assigning more of the graphing region to be devoted to marginal seats.

Value

The function performs the requested plots and exits silently with `invisible{NULL}`.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

`density`, `rug`

Examples

```
data(ca2006)
x <- ca2006$D/(ca2006$D+ca2006$R)
sv <- seatsVotes(x,
                 desc="Democratic Vote Shares, California 2006 congressional elections")

plot(sv)
plot(sv,type="density")
plot(sv,type="density",transform=TRUE)
```

<code>postProcess</code>	<i>remap MCMC output via affine transformations</i>
--------------------------	---

Description

Remap the MCMC iterates in an `ideal` object via an affine transformation, imposing identifying restrictions ex post (aka post-processing).

Usage

```
postProcess(object, constraints, debug = FALSE)
```

Arguments

<code>object</code>	an object of class <code>ideal</code>
<code>constraints</code>	list of length $d+1$, each component providing a set of d restrictions, where d is the dimension of the fitted <code>ideal</code> model. The name of each component should uniquely match a legislator/subject's name. See Details.
<code>debug</code>	logical flag for verbose output, used for debugging

Details

Item-response models are unidentified without restrictions on the underlying parameters. Consider the $d=1$ dimensional case. The model is

$$P(y_{ij} = 1) = F(x_i\beta_j - \alpha_j)$$

Any linear transformation of the latent traits, say,

$$x^* = mx + c$$

can be exactly offset by applying the appropriate linear transformations to the item/bill parameters, meaning that there is no unique set of values for the model parameters that will maximize the likelihood function. In higher dimensions, the latent traits can also be transformed via any arbitrary rotation, dilation and translation, with offsetting transformations applied to the item/bill parameters.

One strategy in MCMC is to ignore the lack of identification at run time, but apply identifying restrictions ex post, “post-processing” the MCMC output, iteration-by-iteration. In a d -dimensional IRT model, a sufficient condition for global identification is to fix $d+1$ latent traits, provided the constrained latent traits span the d dimensional latent space. This function implements this strategy. The user supplies a set of constrained ideal points in the `constraints` list. The function then processes the MCMC output in the `ideal` object, finding the transformation that maps the current iteration's sampled values for x (latent traits/ideal points) into the sub-space of identified parameters defined by the fixed points in `constraints`; i.e., what is the affine transformation that maps the unconstrained ideal points into the constraints. Aside from miniscule numerical inaccuracies resulting from matrix inversion etc, this transformation is exact: after post-processing, the $d+1$ constrained points do not vary over the MCMC iterations. The remaining $n-d-1$ ideal points are subject to (posterior) uncertainty; the “random tour” of the joint parameter space of these parameters produced by the MCMC algorithm has been mapped into a subspace in which the parameters are globally identified.

If the `ideal` object was produced with `store.item` set to `TRUE`, then the item parameters are also post-processed, applying the inverse transformation. Specifically, recall that the IRT model is

$$P(y_{ij} = 1) = F(x'_i \beta_j)$$

where in this formulation x_i is a vector of length $d+1$, including a “1” to put a constant term into the model (i.e., the intercept or “minus-difficulty” parameter is part of β_j). Let A denote the non-singular, $d+1$ -by- $d+1$ matrix that maps the x into the space of identified parameters. Recall that this transformation is computed iteration by iteration. Then each x_i is transformed to $x_i^* = Ax_i$ and β_j is transformed to $\beta_j^* = A^{-1}\beta_j$, $i = 1, \dots, n$; $j = 1, \dots, m$.

Value

An object of class `ideal`, with components suitably transformed and recomputed (i.e., `x` is transformed and `xbar` recomputed, and if the `ideal` object was fit with `store.item=TRUE`, `beta` is transformed and `betabar` is recomputed).

Note

Applying these transformations to obtain identification can sometimes lead to surprising results. Each data point makes the same likelihood contributions with either the identified or unidentified parameters. But, in general, predictions generated with the parameters set to their posterior means will differ depending on whether one uses the identified subset of parameters or the unidentified parameters. For this reason, caution should be used when using a function such as `predict` after post-processing output from `ideal`. A better strategy is to compute the estimand of interest at each iteration and then take averages over iterations.

When specifying a value of `burnin` different from that used in fitting the `ideal` object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the n -th iteration stored in an `ideal` object will not be iteration n if the user specified `thin>1` in the call to `ideal`. Here, iterations are tagged with their iteration number. Thus, if the user called `ideal` with `thin=10` and `burnin=100` then the stored iterations are numbered 100, 110, 120, Any future subsetting via a `burnin` refers to this iteration number.

Author(s)

Simon Jackman (jackman@stanford.edu)

References

Hoff, Peter, Adrian E. Raftery and Mark S. Handcock. 2002. Latent Space Approaches to Social Network Analysis. *Journal of the American Statistical Association* 97:1090–1098.

Edwards, Yancy D. and Greg M. Allenby. 2003. Multivariate Analysis of Multiple Response Data. *Journal of Marketing Research* 40:321–334.

Rivers, Douglas. 2003. “Identification of Multidimensional Item-Response Models.” Typescript. Department of Political Science, Stanford University.

Examples

```
data(s109)

## short run for examples
idl <- ideal(s109,
            d=1,
            meanzero=FALSE,      ## no identification...!
```

```

        store.item=TRUE,
        maxiter=500,
        burnin=0,
        thin=10)

idlpp <- postProcess(id1,
                    constraints=list(BOXER=-1, INHOFE=1))

## Not run:
id2 <- ideal(s109,
            d=2,
            store.item=TRUE,
            maxiter=1e4,
            burnin=0,
            thin=25)

tracex(id2,d=1:2,
       legis=c("BOXER", "INHOFE", "BYRD", "CHAFEE", "MCCAIN"),
       showAll=TRUE)

id2pp <- postProcess(id2,
                    constraints=list(BOXER=c(-1,0),
                                    INHOFE=c(1,0),
                                    CHAFEE=c(0,.25)))

tracex(id2pp,d=1:2,
       legis=c("BOXER", "INHOFE", "COLLINS", "FEINGOLD", "COLEMAN",
               "CHAFEE", "MCCAIN", "KYL"),
       showAll=TRUE)
## End(Not run)

```

predict.hurdle

Methods for hurdle Objects

Description

Methods for extracting information from fitted hurdle regression model objects of class "hurdle".

Usage

```

## S3 method for class 'hurdle':
predict(object, newdata,
       type = c("response", "prob"), na.action = na.pass, ...)
## S3 method for class 'hurdle':
residuals(object, type = c("pearson", "response"), ...)

## S3 method for class 'hurdle':
coef(object, model = c("full", "count", "zero"), ...)
## S3 method for class 'hurdle':
vcov(object, model = c("full", "count", "zero"), ...)

## S3 method for class 'hurdle':
terms(x, model = c("count", "zero"), ...)

```

```
## S3 method for class 'hurdle':
model.matrix(object, model = c("count", "zero"), ...)
```

Arguments

<code>object, x</code>	an object of class "hurdle" as returned by <code>hurdle</code> .
<code>newdata</code>	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
<code>type</code>	character specifying the type of predictions or residuals, respectively.
<code>na.action</code>	function determining what should be done with missing values in <code>newdata</code> . The default is to predict NA.
<code>model</code>	character specifying for which component of the model the terms or model matrix should be extracted.
<code>...</code>	currently not used.

Details

A set of standard extractor functions for fitted model objects is available for objects of class "hurdle", including methods to the generic functions `print` and `summary` which print the estimated coefficients along with some further information. The `summary` in particular supplies partial Wald tests based on the coefficients and the covariance matrix (estimated from the Hessian in the numerical optimization of the log-likelihood). As usual, the `summary` method returns an object of class "summary.hurdle" containing the relevant summary statistics which can subsequently be printed using the associated `print` method.

The methods for `coef` and `vcov` by default return a single vector of coefficients and their associated covariance matrix, respectively, i.e., all coefficients are concatenated. By setting the `model` argument, the estimates for the corresponding model component can be extracted.

Both the `fitted` and `predict` methods can compute fitted responses. The latter additionally provides fitted probabilities, both for the original and for new data. The `residuals` method can compute raw residuals (observed - fitted) and Pearson residuals (raw residuals scaled by square root of fitted).

The `terms` and `model.matrix` extractors can be used to extract the relevant information for either component of the model.

A `logLik` method is provided, hence `AIC` can be called to compute information criteria.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

See Also

`hurdle`

Examples

```
data("bioChemists", package = "pscl")
fm <- hurdle(art ~ ., data = bioChemists)

plot(residuals(fm) ~ fitted(fm))

coef(fm)
```

```
coef(fm, model = "zero")

summary(fm)
logLik(fm)
```

predict.ideal *predicted probabilities from an ideal object*

Description

Compute predicted probabilities from an `ideal` object. This predict method uses the posterior mean values of x and β to make predictions.

Usage

```
## S3 method for class 'ideal':
predict(object,
        cutoff=.5,
        burnin=NULL,
        ...)

## S3 method for class 'predict.ideal':
print(x,digits=2,...)
```

Arguments

<code>object</code>	an object of class <code>ideal</code> (produced by <code>ideal</code>) with item parameters (beta) stored.
<code>cutoff</code>	numeric, a value between 0 and 1, the threshold to be used for separating predicted probabilities into yea and nay votes.
<code>burnin</code>	of the recorded MCMC samples, how many to discard as burnin? Default is NULL, in which case the value of <code>burnin</code> in the <code>ideal</code> object is used.
<code>x</code>	object of class <code>predict.ideal</code>
<code>digits</code>	number of digits in printed object
<code>...</code>	further arguments passed to or from other methods.

Details

Predicted probabilities are computed using the mean of the posterior density of of

$$x$$

(ideal points, or latent ability) and

$$beta$$

(bill or item parameters). The percentage correctly predicted are determined by counting the percentages of votes with predicted probabilities of a Yea vote greater than or equal to the `cutoff` as the threshold.

Value

An object of class `predict.ideal`, containing:

<code>pred.probs</code>	the calculated predicted probability for each legislator for each vote.
<code>prediction</code>	the calculated prediction (0 or 1) for each legislator for each vote.
<code>correct</code>	for each legislator for each vote, whether the prediction was correct.
<code>legis.percent</code>	for each legislator, the percent of votes correctly predicted.
<code>vote.percent</code>	for each vote, the percent correctly predicted.
<code>yea.percent</code>	the percent of yea votes correctly predicted.
<code>nay.percent</code>	the percent of nay votes correctly predicted.
<code>party.percent</code>	the average value of the percent correctly predicted by legislator, separated by party, if party information exists in the <code>rollcall</code> object used for <code>ideal</code> . If no party information is available, <code>party.percent = NULL</code> .
<code>overall.percent</code>	the total percent of votes correctly predicted.
<code>ideal</code>	the name of the <code>ideal</code> object, which can be later <code>evaluated</code>
<code>desc</code>	string, the descriptive text from the <code>rollcall</code> object passed to <code>ideal</code>

Note

When specifying a value of `burnin` different from that used in fitting the `ideal` object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the `n`-th iteration stored in an `ideal` object will not be iteration `n` if the user specified `thin>1` in the call to `ideal`. Here, iterations are tagged with their iteration number. Thus, if the user called `ideal` with `thin=10` and `burnin=100` then the stored iterations are numbered 100, 110, 120, Any future subsetting via a `burnin` refers to this iteration number.

See Also

[ideal](#), [summary.ideal](#), [plot.predict.ideal](#)

Examples

```
data(s109)

## Not run:
idl <- ideal(s109, meanzero=TRUE,
            store.item=TRUE)      ## too long for examples
## End(Not run)

idl <- ideal(s109,
            d=1,
            meanzero=TRUE,
            store.item=TRUE,      ## need this to be TRUE for predict
            maxiter=500,
            burnin=100,
            thin=10)

phat <- predict(idl)
phat      ## print method
```

predict.zeroinfl *Methods for zeroinfl Objects*

Description

Methods for extracting information from fitted zero-inflated regression model objects of class "zeroinfl".

Usage

```
## S3 method for class 'zeroinfl':
predict(object, newdata,
        type = c("response", "prob"), na.action = na.pass, ...)
## S3 method for class 'zeroinfl':
residuals(object, type = c("pearson", "response"), ...)

## S3 method for class 'zeroinfl':
coef(object, model = c("full", "count", "zero"), ...)
## S3 method for class 'zeroinfl':
vcov(object, model = c("full", "count", "zero"), ...)

## S3 method for class 'zeroinfl':
terms(x, model = c("count", "zero"), ...)
## S3 method for class 'zeroinfl':
model.matrix(object, model = c("count", "zero"), ...)
```

Arguments

object, x	an object of class "zeroinfl" as returned by zeroinfl .
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
type	character specifying the type of predictions or residuals, respectively.
na.action	function determining what should be done with missing values in newdata. The default is to predict NA.
model	character specifying for which component of the model the terms or model matrix should be extracted.
...	currently not used.

Details

A set of standard extractor functions for fitted model objects is available for objects of class "zeroinfl", including methods to the generic functions [print](#) and [summary](#) which print the estimated coefficients along with some further information. The [summary](#) in particular supplies partial Wald tests based on the coefficients and the covariance matrix (estimated from the Hessian in the numerical optimization of the log-likelihood). As usual, the [summary](#) method returns an object of class "summary.zeroinfl" containing the relevant summary statistics which can subsequently be printed using the associated [print](#) method.

The methods for [coef](#) and [vcov](#) by default return a single vector of coefficients and their associated covariance matrix, respectively, i.e., all coefficients are concatenated. By setting the [model](#) argument, the estimates for the corresponding model components can be extracted.

Both the `fitted` and `predict` methods can compute fitted responses. The latter additionally provides fitted probabilities, both for the original and for new data. The `residuals` method can compute raw residuals (observed - fitted) and Pearson residuals (raw residuals scaled by square root of fitted).

The `terms` and `model.matrix` extractors can be used to extract the relevant information for either component of the model.

A `logLik` method is provided, hence `AIC` can be called to compute information criteria.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

See Also

`zeroinfl`

Examples

```
data("bioChemists", package = "pscl")

fm_zip <- zeroinfl(art ~ ., data = bioChemists)
plot(residuals(fm_zip) ~ fitted(fm_zip))

coef(fm_zip)
coef(fm_zip, model = "count")

summary(fm_zip)
logLik(fm_zip)
```

predprob

compute predicted probabilities from fitted models

Description

Compute predicted probabilities from fitted models, optionally at new covariate values.

Usage

```
predprob(obj, ...)
```

Arguments

<code>obj</code>	fitted model object
<code>...</code>	other arguments

Details

See documentation for specific methods.

Value

A matrix of predicted probabilities, each row a vector of predicted probabilities over the range of responses seen in the data (i.e., $\min(Y) : \max(Y)$), conditional on the values of covariates.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[predprob.glm](#), [predprob.zeroinfl](#)

Examples

```
data("bioChemists")
zip <- zeroinfl(art ~ . | ., data = bioChemists, EM = TRUE)
phat <- predprob(zip)

newdata <- expand.grid(list(fem="Men",mar="Married",
                          kid5=1,phd=3.103,
                          ment=0:77))
phat <- predprob(zip, newdata = newdata)
```

predprob.glm

Predicted Probabilities for GLM Fits

Description

Obtains predicted probabilities from a fitted generalized linear model object.

Usage

```
## S3 method for class 'glm':
predprob(obj,newdata=NULL,...)
```

Arguments

obj	a fitted object of class inheriting from "glm"
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
...	arguments passed to or from other methods

Details

This method is only defined for glm objects with `family=binomial` or `family=poisson`, or negative binomial count models fit with the `glm.nb` function in `library(MASS)`.

Value

a matrix of predicted probabilities. Each row in the matrix is a vector of probabilities, assigning predicted probabilities over the range of responses actually observed in the data. For instance, for models with `family=binomial`, the matrix has two columns for the "zero" (or failure) and "one" (success) outcomes, respectively, and trivially, each row in the matrix sums to 1.0. For counts fit with `family=poisson` or via `glm.nb`, the matrix has `length(min(y):max(y))` columns. Each observation used in fitting the model generates a row to the returned matrix; alternatively, if `newdata` is supplied, the returned matrix will have as many rows as in `newdata`.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[predict.glm](#)

Examples

```
data(bioChemists)
glm1 <- glm(art ~ .,
            data=bioChemists,
            family=poisson,
            trace=TRUE) ## poisson GLM
phat <- predprob(glm1)
apply(phat,1,sum)      ## almost all 1.0
```

predprob.ideal	<i>predicted probabilities from fitting ideal to rollcall data</i>
----------------	--

Description

Computes predicted probabilities of a “Yea” vote conditional on the posterior means of the legislators’ ideal points and vote-specific parameters.

Usage

```
## S3 method for class 'ideal':
predprob(obj, ...)
```

Arguments

obj	An object of class ideal
...	Arguments to be passed to other functions

Details

This is a wrapper function to [predict.ideal](#), extracting just the predicted probabilities component of the object returned by that function. Predicted probabilities can and are generated for each voting decision, irrespective of whether the legislator actually voted on any particular roll call.

Value

A [matrix](#) of dimension n (number of legislators) by m (number of roll call votes).

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[ideal](#), [predprob](#), [predict.ideal](#)

Examples

```
data(s109)
idl <- ideal(s109,
            d=1,
            meanzero=TRUE,
            store.item=TRUE,
            maxiter=500,
            burnin=100,
            thin=10)
phat <- predprob(idl)
dim(phat)
```

prussian

Prussian army horse kick data

Description

Deaths by year, by corp, from horse kicks.

Usage

```
data(prussian)
```

Format

A data frame with 280 observations on the following 3 variables.

y a numeric vector, count of deaths

year a numeric vector, 18XX, year of observation

corp a [factor](#), corp of Prussian Army generating observation

Source

von Bortkiewicz, L. 1898. *Das Gesetz der Kleinen Zahlen*. Leipzig: Teubner.

Examples

```
data(prussian)
corpP <- glm(y ~ corp, family=poisson,data=prussian)
summary(corpP)
```

readKH	<i>read roll call data in Poole-Rosenthal KH format</i>
--------	---

Description

Creates a `rollcall` object from the flat file format for roll call data used by Keith Poole and Howard Rosenthal.

Usage

```
readKH(file,
        dtl=NULL,
        yea=c(1,2,3),
        nay=c(4,5,6),
        missing=c(7,8,9),
        notInLegis=0,
        desc=NULL,
        debug=FALSE)
```

Arguments

<code>file</code>	string, name of a file or URL holding KH data
<code>dtl</code>	string, name of a file or URL holding KH <code>dtl</code> file (information about votes); default is <code>NULL</code> , indicating no <code>dtl</code> file
<code>yea</code>	numeric, possibly a vector, code(s) for a Yea vote in the rollcall context (or a correct answer in the educational testing context). Default is <code>c(1,2,3)</code> , which corresponds to Yea, Paired Yea, and Announced Yea in Poole/Rosenthal data files.
<code>nay</code>	numeric, possibly a vector, code(s) for a Nay vote in the rollcall context (or an incorrect answer in the educational testing context). Default is <code>c(4,5,6)</code> , which corresponds to Announced Nay, Paired Nay, and Nay in Poole/Rosenthal data files.
<code>missing</code>	numeric and/or <code>NA</code> , possible a vector, code(s) for missing data. Default is <code>c(0,7,8,9,NA)</code> ; the first four codes correspond to Not Yet a Member, Present (some Congresses), Present (some Congresses), and Not Voting.
<code>notInLegis</code>	numeric or <code>NA</code> , possibly a vector, code(s) for the legislator not being in the legislature when a particular roll call was recorded (e.g., deceased, retired, yet to be elected). Default is <code>0</code> for Poole/Rosenthal data files.
<code>desc</code>	string, describing the data, e.g., 82nd U.S. House of Representatives; default is <code>NULL</code>
<code>debug</code>	logical, print debugging information for net connection

Details

Keith Poole and Howard Rosenthal have gathered an impressive collection of roll call data, spanning every roll call cast in the United States Congress. This effort continues now as a real-time exercise, via a collaboration with Jeff Lewis (109th Congress onwards). Nolan McCarty collaborated on the compilation of roll call data for the 102nd through 108th Congress.

This function relies on some hard-coded features of Poole-Rosenthal flat files, and assumes that the file being supplied has the following structure (variable, start-end columns):

ICPSR legislator unique ID 4-8

ICPSR state ID 9-10

Congressional District 11-12

state name 13-20

party code 21-23

legislator name 26-36

roll-call voting record 37 to end-of-record

This function reads data files in that format, and creates a `rollcall`, for which there are useful methods such as `summary.rollcall`. The `legis.data` component of the `rollcall` object is a `data.frame` which contains:

state a 2-character string abbreviation of each legislator's state

icpsrState a 2-digit numeric code for each legislator's state, as used by the Inter-university Consortium for Political and Social Research (ICPSR)

cd numeric, the number of each legislator's congressional district within each state; this is always 0 for members of the Senate

icpsrLegis a unique numeric identifier for each legislator assigned by the ICPSR, as corrected by Poole and Rosenthal, see <http://voteview.com/icpsr.htm>

partyName character string, the name of each legislator's political party

party numeric, code for each legislator's political party; see <http://voteview.com/party3.dat>

The `rownames` attribute of this data frame is a concatenation of the legislators' names, party abbreviations (for Democrats and Republicans) and state, and (where appropriate), a district number; e.g., Bonner (R AL-1). This tag is also provided in the `legis.name` component of the returned `rollcall` object.

Poole and Rosenthal also make `dt1` files available for Congresses 1 through 106. These files contain information about the votes themselves, in a multiple-line per vote `ascii` format, and reside in the `dt1` director of Poole's web site, e.g., <ftp://pooleandrosenthal.com/dt1/102s.dt1> is the `dt1` file for the 102nd Senate. The default is to presume that no such file exists. When a `dt1` file is available, and is read, the `votes.data` attribute of the resulting `rollcall` object is a `data.frame` with one record per vote, with the following variables:

date vector of class `Date`, date of the rollcall, if available; otherwise `NULL`

description vector of mode `character`, descriptive text

The `dt1` files are presumed to have the date of the rollcall in the first line of text for each roll call, and lines 3 onwards contain descriptive text.

Finally, note also that the Poole/Rosenthal data sets often include the U.S. President as a pseudo-legislator, adding the announced positions of a president or the administration to the roll call matrix. This adds an extra "legislator" to the data set and can sometimes produce surprising results (e.g., a U.S. Senate of 101 senators), and a "legislator" with a surprisingly low party loyalty score (since the President/administration only announces positions on a relatively small fraction of all Congressional roll calls).

Value

an object of class `rollcall`, with components created using the identifying information in the Poole/Rosenthal files. If the function can not read the file (e.g., the user specified a URL and the machine is not connected to the Internet), the function fails with an error message (set `debug=TRUE` to help resolve these issues).

Author(s)

Simon Jackman (jackman@stanford.edu)

References

Poole, Keith and Howard Rosenthal. 1997. *Congress: A Political-Economic History of Roll Call Voting*. New York: Oxford University Press.

Poole, Keith. <http://voteview.ucsd.edu>

Rosenthal, Howard L. and Keith T. Poole. *United States Congressional Roll Call Voting Records, 1789-1990: Reformatted Data [computer file]*. 2nd ICPSR release. Pittsburgh, PA: Howard L. Rosenthal and Keith T. Poole, Carnegie Mellon University, Graduate School of Industrial Administration [producers], 1991. Ann Arbor, MI: Inter-university Consortium for Political and Social Research [distributor], 2000. <http://webapp.icpsr.umich.edu/cocoon/ICPSR-STUDY/09822.xml>

See Also

`rollcall`

Examples

```
## Not run:
h107 <- readKH("ftp://voteview.com/hou107kh.ord",
              desc="107th U.S. House of Representatives")

s107 <- readKH("ftp://voteview.com/sen107kh.ord",
              desc="107th U.S. Senate")

## Jeff Lewis has quasi-real-time roll call data on his site
## in the Poole/Rosenthal format
s109 <- readKH("http://adric.sscnet.ucla.edu/rollcall/static/S109.ord",
              desc="109th U.S. Senate",
              debug=TRUE)
## End(Not run)
```

`rollcall`

create an object of class rollcall

Description

Create a `rollcall` object, used for the analysis of legislative voting or, equivalently, item-response modeling of binary data produced by standardized tests, etc.

Usage

```
rollcall(data,
         yea=1, nay=0, missing=NA, notInLegis=9,
         legis.names=NULL, vote.names=NULL,
         legis.data=NULL, vote.data=NULL,
         desc=NULL, source=NULL)
```

Arguments

<code>data</code>	voting decisions (for roll calls), or test results (for IRT). Can be in one of two forms. First, <code>data</code> may be a <code>matrix</code> , with rows corresponding to legislators (subjects) and columns to roll calls (test items). <code>data</code> can also be a <code>list</code> with an element named <code>votes</code> containing the matrix described above.
<code>yea</code>	numeric, possibly a vector, code(s) for a Yea vote in the rollcall context, or a correct answer in the educational testing context. Default is 1.
<code>nay</code>	numeric, possibly a vector, code(s) for a Nay vote in the rollcall context, or an incorrect answer in the educational testing context. Default is 0.
<code>missing</code>	numeric or NA, possibly a vector, code(s) for missing data. Default is NA.
<code>notInLegis</code>	numeric or NA, possibly a vector, code(s) for the legislator not being in the legislature when a particular roll call was recorded (e.g., deceased, retired, yet to be elected).
<code>legis.names</code>	a vector of names of the legislators or individuals. If <code>data</code> is a list and contains an element named <code>legis.names</code> , then the list element will be used. Names will be generated if not supplied.
<code>vote.names</code>	a vector of names or labels for the votes or items. If <code>data</code> is a list and contains an element named <code>vote.names</code> , then the list element will be used. Names will be generated if not supplied.
<code>legis.data</code>	a <code>matrix</code> or <code>data.frame</code> containing covariates specific to each legislator/test-taker; e.g., party affiliation, district-level covariates. If this object does not have the same number of rows as <code>data</code> , an error is returned.
<code>vote.data</code>	a <code>matrix</code> or <code>data.frame</code> containing covariates specific to each roll call vote or test item; e.g., a timestamp, the bill sponsor, descriptive text indicating the type of vote. If this object does not have the same number of row as the number of columns in <code>data</code> , an error is returned.
<code>desc</code>	character, a string providing an (optional) description of the data being used. If <code>data</code> is a list and contains an element named <code>desc</code> , then this will be used.
<code>source</code>	character, a string providing an (optional) description of where the roll call data originated (e.g., a URL or a short-form reference). Used in print and summary methods.

Details

See below for methods that operate on objects of class `rollcall`.

Value

An object of class `rollcall`, a list with the following components:

votes	a matrix containing voting decisions, with rows corresponding to legislators (test subjects) and columns to roll call votes (test items). Legislators (test subjects) and items (or votes) have been labeled in the dimnames attribute of this matrix, using the <code>legis.names</code> and/or <code>vote.names</code> arguments, respectively.
codes	a list with named components <code>yea</code> , <code>nay</code> , <code>notInLegis</code> and <code>missing</code> , each component a numeric vector (possibly of length 1 and possibly NA), indicating how entries in the <code>votes</code> component of the <code>rollcall</code> object should be considered. This list simply gathers up the values in the <code>yea</code> , <code>nay</code> , <code>notInLegis</code> and <code>missing</code> arguments passed to the function.
n	numeric, number of legislators, equal to <code>dim(votes)[1]</code>
m	numeric, number of votes, equal to <code>dim(votes)[2]</code>
legis.data	user-supplied data on legislators/test-subjects.
vote.data	user-supplied data on rollcall votes/test-items.
desc	any user-supplied description. If no description was provided, defaults <code>desc</code> defaults to NULL.
source	any user-supplied source information (e.g., a url or a short-form reference). If no description is provided, <code>source</code> defaults to NULL.

See Also

[readKH](#) for creating objects from files (possibly over the web), in the format used for data from the United States Congress used by Keith Poole and Howard Rosenthal (and others).

[summary.rollcall](#), [ideal](#) for model fitting.

Examples

```
## generate some fake roll call data
set.seed(314159265)
fakeData <- matrix(sample(x=c(0,1),size=5000,replace=TRUE),
                   50,100)
rc <- rollcall(fakeData)
is(rc,"rollcall")      ## TRUE
rc                     ## print the rollcall object on screen

data(sc9497)           ## Supreme Court example data
rc <- rollcall(data=sc9497$votes,
               legis.names=sc9497$legis.names,
               desc=sc9497$desc)
summary(rc,verbose=TRUE)

## Not run:
## s107
## could use readKH for this
dat <- readLines("sen107kh.ord")
dat <- substring(dat,37)
mat <- matrix(NA,ncol=nchar(dat[1]),nrow=length(dat))
for(i in 1:103){
  mat[i,] <- as.numeric(unlist(strsplit(dat[i],
                                       split=character(0))))
}

s107 <- rollcall(mat,
```

```

        yea=c(1,2,3),
        nay=c(4,5,6),
        missing=c(7,8,9),
        notInLegis=0,
        desc="107th U.S. Senate",
        source="http://voteview.ucsd.edu")
summary(s107)
## End(Not run)

```

s109

rollcall object, 109th U.S. Senate

Description

A sample rollcall object, generated using Jeff Lewis' quasi-real-time collection of the rollcalls of the 109th U.S. Senate (2005-2006).

Usage

```
data(s109)
```

Format

A `rollcall` object containing the recorded votes of the 109th U.S. Senate, plus information identifying the legislators and the rollcalls.

Details

The most recent rollcall in this version of the data set is dated June 29, 2006.

Source

Jeff Lewis' web site: <http://adric.sscnet.ucla.edu/rollcall/static/S109.ord>

Information identifying the votes is available at <http://adric.sscnet.ucla.edu/rollcall/static/S109desc.csv>

Examples

```

data(s109)
require(psc1)
is(s109,"rollcall")
s109
summary(s109)
summary(s109,verbose=TRUE)
## Not run:
## how s109 was created
s109 <- readKH("http://adric.sscnet.ucla.edu/rollcall/static/S109.ord",
              desc="109th U.S. Senate",
              debug=TRUE)
url <- "http://adric.sscnet.ucla.edu/rollcall/static/S109desc.csv"
s109$vote.data <- data.frame(read.csv(file=url,header=TRUE))
s109$vote.data$date <- as.Date(s109$vote.data$date,

```

```

                                format="
dimnames(s109$votes)[[2]] <- paste(s109$vote.data$session,
                                s109$vote.data$number, sep="-")
## End(Not run)

```

sc9497

votes from the United States Supreme Court, from 1994-1997

Description

This data set provides information on the United States Supreme Court from 1994-1997. Votes included are non-unanimous.

Usage

```
data(sc9497)
```

Format

A list containing the elements:

votes a matrix of the votes, 0=Nay, 1=Yea, NA=Abstained or missing data. The matrix columns are labeled with `vote.names` and the rows are labeled with `legis.names`.

legis.names a vector of the names of the nine Justices sitting on the court at this time.

party NULL; exists for consistency with House and Senate data sets.

state NULL; exists for consistency with House and Senate data sets.

district NULL; exists for consistency with House data sets.

id NULL; exists for consistency with House and Senate data sets.

vote.names a vector of strings numbering the cases simply to distinguish them from one another.

desc a description of the data set.

Source

Harold J. Spaeth (1999). *United States Supreme Court Judicial Database, 1953-1997 Terms*. Ninth edition. Inter-university Consortium for Political and Social Research. Ann Arbor, Michigan. <http://www.icpsr.umich.edu/>

seatsVotes

A class for creating seats-votes curves

Description

Convert a vector of vote shares into a seats-vote curve object, providing estimates of partisan bias.

Usage

```
seatsVotes(x, desc = NULL, method = "uniformSwing")
```

Arguments

<code>x</code>	a vector of vote shares for a specific party (either proportions or percentages)
<code>desc</code>	descriptive text
<code>method</code>	how to simulate a seats-vote curve; the only supported method at this stage is <code>uniformSwing</code> .

Details

Simulation methods are required to induce a seats-votes curve given a vector of vote shares from one election. The uniform swing method simply slides the empirical distribution function of the vote shares “up” and “down”, computing the proportion of the vote shares that lie above .5 (by construction, the winning percentage in a two-party election) for each new location of the vector of vote shares. That is, as the empirical CDF of the observed vote shares slides up or down, more or less seats cross the .5 threshold. A seats-votes curve is formed by plotting the seat share above .5 as a function of the average district-level vote share (a weakly monotone function, since the empirical CDF constitutes a set of sufficient statistics for this problem). The simulation is run so as to ensure that average district-level vote shares range between 0 and 1.

The extent to which the seats-votes curve departs from symmetry is known as bias. More specifically, the vertical displacement of the seats-votes curve from .5 when average district-level vote share is .5 is conventionally reported as an estimate of the bias of the electoral system.

Different methods produce different estimates of seats-votes curves and summary estimands such as bias. The uniform swing method is completely deterministic and does not produce any uncertainty assessment (e.g., confidence intervals etc).

Value

An object of class `seatsVotes`, with components

<code>s</code>	Estimated seat shares over the range of simulated average, district-level vote shares
<code>v</code>	Simulated average district-level vote shares
<code>x</code>	observed seat shares, with missing data removed
<code>desc</code>	user-supplied descriptive character string
<code>call</code>	a list of class <code>call</code> , the call to the function

Note

Additional methods to come later.

Author(s)

Simon Jackman (jackman@stanford.edu)

References

- Tufte, Edward R. 1973. The Relationship Between Seats and Votes in Two-Party Systems. *American Political Science Review*. 67(2):540-554.
- Gelman, Andrew and Gary King. 1990. Estimating the Consequences of Electoral Redistricting. *Journal of the American Statistical Association*. 85:274-282.
- Jackman, Simon. 1994. Measuring Electoral Bias: Australia, 1949-93. *British Journal of Political Science*. 24(3):319-357.

See Also

[plot.seatsVotes](#) for plotting methods.

Examples

```
data(ca2006)
x <- ca2006$D/(ca2006$D+ca2006$R)
sv <- seatsVotes(x,
                 desc="Democratic Vote Shares, California 2006 congressional elections")
```

state.info

information about the American states needed for U.S. Congress

Description

Numeric codes and names of 50 states and the District of Columbia, required to parse Keith Poole and Howard Rosenthal's collections of U.S. Congressional roll calls.

Usage

```
data(state.info)
```

Format

`icpsr` integer, numeric code for state used by the Inter-university Consortium for Political and Social Research

`state` character, name of state or Washington D.C.

`year` numeric or NA, year of statehood

Details

The function [readKH](#) converts the integer ICPSR codes into strings, via a table lookup in this data frame. Another table lookup in [state.abb](#) provides the 2-letter abbreviation commonly used in identifying American legislators, e.g., KENNEDY, E (D-MA).

Source

Various ICPSR codebooks. <http://www.icpsr.umich.edu>

See Also[state](#)

summary.ideal	<i>summary of an ideal object</i>
---------------	-----------------------------------

Description

Provides a summary of the output from ideal point estimation contained in an object of class `ideal`.

Usage

```
## S3 method for class 'ideal':
summary(object, quantiles = c(.025, .975),
        burnin=NULL,
        include.beta=FALSE,...)
```

Arguments

<code>object</code>	an object of class <code>ideal</code> .
<code>quantiles</code>	a list of quantiles to report for each legislator's ideal point and each item's discrimination parameter (if stored in the <code>ideal</code> object).
<code>burnin</code>	of the recorded MCMC samples, how many to discard as burnin? Default is <code>NULL</code> , in which case the value of <code>burnin</code> in the <code>ideal</code> object is used.
<code>include.beta</code>	whether or not to calculate summary statistics of beta, if beta is available. If the item parameters were not stored in the <code>ideal</code> object, then <code>include.beta</code> is ignored.
<code>...</code>	further arguments passed to or from other functions

Details

The tests of whether a discrimination parameters are distinguishable from zero first checks to see if the two most extreme `quantiles` are symmetric (e.g., as are the default value of `.025` and `.975`). If so, the corresponding quantiles of the MCMC samples for each discrimination parameter are inspected to see if they have the same sign. If they do, then the corresponding discrimination parameter is flagged as distinguishable from zero; otherwise not.

Value

An item of class `summary.ideal` with elements:

<code>object</code>	the name of the ideal object as an unevaluated expression , produced by <code>match.call()\$object</code>
<code>xResults</code>	a list of length <code>d</code> (the dimension of the fitted model). Component <code>i</code> of the list is a matrix summarizing the MCMC output for the <code>n</code> legislators' ideal points on the <code>i</code> -th dimension of the model. The columns of this matrix contain the mean of the MCMC draws from the posterior density of the legislators ideal points, the standard deviation, and the requested <code>quantiles</code> .

bResults	a list of length $d+1$, similar to <code>xResults</code> , but containing summaries of the bill parameters; i.e., there are d discrimination parameters per bill, plus an intercept. If the bill/item parameters were not stored when <code>ideal</code> was called (<code>store.item=FALSE</code>), or <code>include.beta=FALSE</code> , then <code>bResults</code> is a list of length zero.
bSig	a <code>link{list}</code> of length d , each component a vector of length m , of mode <code>logical</code> , equal to <code>TRUE</code> if the corresponding discrimination parameter is distinguishable from zero; see Details . If <code>store.item</code> was set to <code>FALSE</code> when <code>ideal</code> was invoked, then <code>bSig</code> is a list of length zero.
party.quant	if party information is available through the <code>rollcall</code> object that was used to run <code>ideal</code> , then <code>party.quant</code> gives the posterior mean of the legislators' ideal points by party, by dimension. If no party information is available, then <code>party.quant=NULL</code> .

Note

When specifying a value of `burnin` different from that used in fitting the `ideal` object, note a distinction between the iteration numbers of the stored iterations, and the number of stored iterations. That is, the n -th iteration stored in an `ideal` object will not be iteration n if the user specified `thin>1` in the call to `ideal`. Here, iterations are tagged with their iteration number. Thus, if the user called `ideal` with `thin=10` and `burnin=100` then the stored iterations are numbered 100, 110, 120, ... Any future subsetting via a `burnin` refers to this iteration number.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[ideal](#)

Examples

```
## fake example
set.seed(314159265)
fakeData <- matrix(sample(x=c(0,1),size=1000,replace=TRUE),
                    10,100)
rc <- rollcall(fakeData)
## short-run for demo purposes
idFake <- ideal(rc,maxiter=500,burnin=100,thin=10)
summary(idFake)

## Supreme Court Example
data(sc9497)
rc <- rollcall(data=sc9497$votes,
               legis.names=sc9497$legis.names,
               desc=sc9497$desc)
id1 <- ideal(rc)
summary(id1)

## Not run:
data(s109)
cl2 <- constrain.legis(s109,
                       x=list("KENNEDY (D MA)"=c(-1,0),
                              "ENZI (R WY)"=c(1,0),
```

```

        "CHAFEE (R RI)"=c(0,-.5)),
        d=2)
id2Constrained <- ideal(s109,
        d=2,
        priors=c12,      ## priors (w constraints)
        startvals=c12,  ## start value (w constraints)
        store.item=TRUE,
        maxiter=5000,
        burnin=500,
        thin=25)

summary(id2Constrained,
        include.items=TRUE)
## End(Not run)

```

summary.rollcall *summarize a rollcall object*

Description

Provides a summary of the information about votes, legislators, etc in a [rollcall](#) object.

Usage

```

## S3 method for class 'rollcall':
summary(object,
        dropList=list(codes = "notInLegis", lop = 0),
        verbose=FALSE,...)

## S3 method for class 'summary.rollcall':
print(x, digits=1, ...)

```

Arguments

object	an rollcall object.
dropList	a list or alist , listing voting decisions, legislators and/or votes to be dropped from the summary; see dropRollCall for details.
verbose	logical, if TRUE, compute legislator-specific and vote-specific Yea/Nay/NA summaries
x	an object of class <code>summary.rollcall</code>
digits	number of decimal places in printed display
...	further arguments passed to or from other methods.

Value

An object of class `summary.rollcall` with the following elements (depending on the logical flag `verbose`):

n	number of legislators in the rollcall object, after processing the <code>dropList</code>
m	number of roll call votes in the rollcall object, after processing the <code>dropList</code>

codes	a list that describes how the voting decisions in the rollcall matrix (<code>object\$votes</code>) map into “Yea” and “Nay” etc, after processing the <code>dropList</code> ; see rollcall for more details
allVotes	a matrix containing a tabular breakdown of all votes in the rollcall matrix (<code>object\$votes</code>), after processing the <code>dropList</code>
partyTab	a tabular breakdown of the legislators’ party affiliations, after processing the <code>dropList</code> , and only if party affiliations are supplied as <code>object\$legis.data\$party</code> ; see rollcall for details
lopSided	a tabular summary of the frequency of lop-sided roll call votes in the rollcall object, again, after processing the <code>dropList</code>
legisTab	a tabular summary of each legislators’ voting history
partyLoyalty	the proportion of times that each legislator votes the way that a majority of his or her fellow partisans did, provided party affiliations are available
voteTab	a tabular summary of each rollcall’s votes
call	the matched call used to invoke <code>summary.rollcall</code>

See Also

[rollcall](#)

Examples

```
set.seed(314159265)
fakeData <- matrix(sample(x=c(0,1),size=1000,replace=TRUE),
                    10,100)
rc <- rollcall(fakeData)
rc

data(sc9497)
rc <- rollcall(sc9497)
summary(rc)

data(s109)
summary(s109)
summary(s109,verbose=TRUE)
```

tracex	<i>trace plot of MCMC iterates, posterior density of legislators’ ideal points</i>
--------	--

Description

Produces a trace plot of the MCMC samples from the posterior density of legislators’ [ideal](#) points.

Usage

```
tracex(object, legis=NULL, d=1, conf.int=0.95,
        showAll = FALSE, burnin=NULL)
```

Arguments

<code>object</code>	an object of class <code>ideal</code> .
<code>legis</code>	a vector of either the names of legislators (or partial matches of the names as given in the <code>dimnames</code> of <code>object\$x</code>).
<code>d</code>	numeric, either a scalar or a vector of length two, the dimension(s) to be traced.
<code>conf.int</code>	numeric, the level of the confidence interval on the posterior mean to be plotted.
<code>showAll</code>	logical, if TRUE and <code>length(d)==2</code> , display traces for all selected legislators' ideal points on the one plot.
<code>burnin</code>	of the recorded MCMC samples, how many to discard as burnin? Default is NULL, in which case the value of <code>burnin</code> in the <code>ideal</code> object is used.

Details

Produces a trace plot showing the history of the MCMC iterations for the ideal point of each of the legislators (partially) named in `legis`. For `d=1`, each trace plot includes a trace over iterations, the cumulative mean, a moving average, the MCMC-based estimate of the mean of the posterior, and a confidence interval (specified by `conf.int`) around the mean of the posterior (using the estimated [quantiles](#)) of the respective MCMC iterates). All of these values are calculated discarding the initial `burnin` iterations.

When `d` is a vector of length two, a 2-dimensional trace plot is displayed, with the `d[1]` dimension on the horizontal axis, and the `d[2]` dimension on the vertical axis.

See Also

[ideal](#); [pmatch](#) for matching legislators' names.

Examples

```
data(s109)
## short run for demo purposes only
id1 <- ideal(s109,
             meanzero=TRUE,
             maxiter=500,
             burnin=100,thin=10)
tracex(id1,legis="KENNEDY")

## n.b., no such legislator has Horrendous Goblin
tracex(id1,legis=c("KENNEDY","BOXER","KYL","Horrendous Goblin"))

## Not run:
id2 <- ideal(s109,
             d=2,
             maxiter=5000, ## unidentified!
             burnin=0,
             thin=50)
tracex(id2,d=1,legis=c("KENNEDY","BOXER","KYL","Horrendous Goblin"))
tracex(id2,d=2,legis=c("KENNEDY","BOXER","KYL","Horrendous Goblin"))
tracex(id2,d=1:2,
       legis=c("KENNEDY","BOXER","KYL","Horrendous Goblin"))

## partial matching
tracex(id2,d=1:2,
       legis=c("KENN","BOX","BID","SNO","SPEC","MCCA","KYL",
```

```

        "Horrendous Goblin" ),
    showAll=TRUE)
## End(Not run)

```

unionDensity	<i>cross national rates of trade union density</i>
--------------	--

Description

Cross-national data on relative size of the trade unions and predictors, in 20 countries. Two of the predictors are highly collinear, and are the source of a debate between Stephens and Wallerstein (1991), later reviewed by Western and Jackman (1994).

Usage

```
data(unionDensity)
```

Format

- `union` numeric, percentage of the total number of wage and salary earners plus the unemployed who are union members, measured between 1975 and 1980, with most of the data drawn from 1979
- `left` numeric, an index tapping the extent to which parties of the left have controlled governments since 1919, due to Wilensky (1981).
- `size` numeric, log of labor force size, defined as the number of wage and salary earners, plus the unemployed
- `concen` numeric, percentage of employment, shipments, or production accounted for by the four largest enterprises in a particular industry, averaged over industries (with weights proportional to the size of the industry) and the resulting measure is normalized such that the United States scores a 1.0, and is due to Pryor (1973). Some of the scores on this variable are imputed using procedures described in Stephens and Wallerstein (1991, 945).

Source

- Pryor, Frederic. 1973. *Property and Industrial Organization in Communist and Capitalist Countries*. Bloomington: Indiana University Press.
- Stephens, John and Michael Wallerstein. 1991. Industrial Concentration, Country Size and Trade Union Membership. *American Political Science Review* 85:941-953.
- Western, Bruce and Simon Jackman. 1994. Bayesian Inference for Comparative Research. *American Political Science Review* 88:412-423.
- Wilensky, Harold L. 1981. Leftism, Catholicism, Democratic Corporatism: The Role of Political Parties in Recent Welfare State Development. In *The Development of Welfare States in Europe and America*, ed. Peter Flora and Arnold J. Heidenheimer. New Brunswick: Transaction Books.

Examples

```

data(unionDensity)
summary(unionDensity)
pairs(unionDensity,
      labels=c("Union\nDensity",
              "Left\nGovernment"),

```

```

      "log Size of\nLabor Force",
      "Economic\nConcentration"),
lower.panel=function(x,y,digits=2){
  r <- cor(x,y)
  par(usr=c(0,1,0,1))
  text(.5,.5,
       format(c(r,0.123456789),digits=digits)[1],
       cex=1.5)
}
)
ols <- lm(union ~ left + size + concen,
         data=unionDensity)
summary(ols)

```

vectorRepresentation

convert roll call matrix to series of vectors

Description

Extract the information in a roll call matrix as a series of vectors with voting decision, a unique identifier for the legislator and a unique identifier for the roll call.

Usage

```
vectorRepresentation(object, dropList = list(codes = c("missing", "notInLegis")))
```

Arguments

`object` an object of class `rollcall`
`dropList` a `dropList`; see `dropRollCall`

Details

It is often the case that roll call matrices are sparse, say, when the roll call matrix has an “overlapping generations” structure; e.g., consider forming data by pooling across a long temporal sequence of legislatures such that relatively few of the legislators in the data set actually vote on any given roll call. In such a case, representing the data as a roll call matrix is not particularly helpful nor efficient, either for data summaries or modeling.

Value

A `matrix` with z rows, where z is the number of non-missing entries in `object$votes`, with ‘missingness’ defined by the `codes` component of the `dropList`. The matrix has 3 columns:

<code>vote</code>	the voting decision, either a 1 if the corresponding element of the roll call matrix <code>object\$votes</code> is in the <code>yea</code> component of <code>object\$codes</code> , or a 0 if the corresponding element of the roll call matrix is in the <code>nay</code> component of <code>object\$codes</code> . Non-missing entries of the roll call matrix are not stored.
<code>i</code>	the row of the roll call matrix <code>object\$votes</code> that supplied the voting decision; i.e., a unique identifier for the legislator generating this vote
<code>j</code>	the column of the roll call matrix <code>object\$votes</code> that supplied the vote; i.e., a unique identifier for the vote.

Author(s)

Simon Jackman (jackman@stanford.edu)

See Also

[rollcall](#)

Examples

```
data(s109)
y <- vectorRepresentation(s109)
apply(y, 2, table, exclude=NULL)
```

vuong

Vuong's non-nested hypothesis test

Description

Compares two models fit to the same data that do not nest via Vuong's non-nested test.

Usage

```
vuong(m1, m2, digits = getOption("digits"))
```

Arguments

m1	model 1, an object inheriting from class glm, negbin or zeroinfl
m2	model 2, as for model 1
digits	significant digits in printed result

Details

The Vuong non-nested test is based on a comparison of the predicted probabilities of two models that do not nest. Examples include comparisons of zero-inflated count models with their non-zero-inflated analogs (e.g., zero-inflated Poisson versus ordinary Poisson, or zero-inflated negative-binomial versus ordinary negative-binomial). A large, positive test statistic provides evidence of the superiority of model 1 over model 2, while a large, negative test statistic is evidence of the superiority of model 2 over model 1. Under the null that the models are indistinguishable, the test statistic is asymptotically distributed standard normal.

The function will fail if the models do not contain identical values in their respective components named y (the value of the response being modeled).

Value

nothing returned, prints the test-statistic and p value and exits silently.

Author(s)

Simon Jackman (jackman@stanford.edu)

References

Vuong, Q.H. 1989. "Likelihood ratio tests for model selection and non-nested hypotheses." *Econometrica*. 57:307-333.

Examples

```
data("bioChemists")
## compare Poisson GLM and ZIP
glm1 <- glm(art ~ ., data = bioChemists, family = poisson)
zip <- zeroinfl(art ~ . | ., data = bioChemists, EM = TRUE)
vuong(glm1, zip)

## compare negbin with zero-inflated negbin
nbl <- glm.nb(art ~ ., data=bioChemists)
zinb <- zeroinfl(art ~ . | ., data = bioChemists, dist = "negbin", EM = TRUE)
vuong(nbl, zinb)
```

zeroinfl

Zero-inflated Count Data Regression

Description

Fit zero-inflated regression models for count data via maximum likelihood.

Usage

```
zeroinfl(formula, data, subset, na.action,
          dist = c("poisson", "negbin", "geometric"),
          link = c("logit", "probit", "cloglog", "cauchit", "log"),
          control = zeroinfl.control(...),
          model = TRUE, y = TRUE, x = FALSE, ...)
```

Arguments

formula	symbolic description of the model, see details.
data, subset, na.action	arguments controlling formula processing via model.frame .
dist	character specification of count model family (a log link is always used).
link	character specification of link function in the binary zero-inflation model (a binomial family is always used).
control	a list of control arguments specified via zeroinfl.control .
model, y, x	logicals. If TRUE the corresponding components of the fit (model frame, response, model matrix) are returned.
...	arguments passed to zeroinfl.control in the default setup.

Details

Zero-inflated count models are two-component mixture models combining a point mass at zero with a proper count distribution. Thus, there are two sources of zeros: zeros may come from both the point mass and from the count component. Usually the count model is a poisson or negative binomial regression (with log link). The geometric distribution is a special case of the negative binomial with size parameter equal to 1. For modeling the unobserved state (zero vs. count), a binary model is used: in the simplest case only with an intercept but potentially containing regressors. For this zero-inflation model, a binomial model with different links can be used, typically logit or probit.

The formula mainly describes the count data model, i.e., $y \sim x1 + x2$ specifies a count data regression where all zero counts have the same probability of belonging to the zero component. This is equivalent to the model $y \sim x1 + x2 \mid 1$, making it more explicit that the zero-inflation model only has an intercept. Additionally, further regressors can be added to the zero-inflation model so that not all zeros have the same probability for belonging to the point mass component or to the count component. A typical formula is $y \sim x1 + x2 \mid z1 + z2$. The regressors in the zero and the count component can be overlapping (or identical).

All parameters are estimated by maximum likelihood using `optim`, with control options set in `zeroinfl.control`. Starting values can be supplied, estimated by the EM (expectation maximization) algorithm, or by `glm.fit` (the default). The latter corresponds to the first iteration of the EM algorithm and initializes the unobserved state as $y > 0$, i.e., all zeros are in the perfect component and only the non-zero counts in the count component. Standard errors are derived numerically using the Hessian matrix returned by `optim`. See `zeroinfl.control` for details.

The returned fitted model object is of class "zeroinfl" and is similar to fitted "glm" objects. For elements such as "coefficients" or "terms" a list is returned with elements for the zero and count component, respectively. For details see below.

A set of standard extractor functions for fitted model objects is available for objects of class "zeroinfl", including methods to the generic functions `print`, `summary`, `coef`, `vcov`, `logLik`, `residuals`, `predict`, `fitted`, `terms`, `model.matrix`. See `predict.zeroinfl` for more details on all methods.

Value

An object of class "zeroinfl", i.e., a list with components including

<code>coefficients</code>	a list with elements "count" and "zero" containing the coefficients from the respective models,
<code>residuals</code>	a vector of raw residuals (observed - fitted),
<code>fitted.values</code>	a vector of fitted means,
<code>optim</code>	a list with the output from the <code>optim</code> call for minimizing the negative log-likelihood,
<code>control</code>	the control arguments passed to the <code>optim</code> call,
<code>start</code>	the starting values for the parameters passed to the <code>optim</code> call,
<code>n</code>	number of observations,
<code>df.null</code>	residual degrees of freedom for the null model ($= n - 2$),
<code>df.residual</code>	residual degrees of freedom for fitted model,
<code>terms</code>	a list with elements "count", "zero" and "full" containing the terms objects for the respective models,
<code>theta</code>	estimate of the additional θ parameter of the negative binomial model (if a negative binomial regression is used),

SE.logtheta	standard error for $\log(\theta)$,
loglik	log-likelihood of the fitted model,
vcov	covariance matrix of all coefficients in the model (derived from the Hessian of the <code>optim</code> output),
dist	character string describing the count distribution used,
link	character string describing the link of the zero-inflation model,
linkinv	the inverse link function corresponding to <code>link</code> ,
converged	logical indicating successful convergence of <code>optim</code> ,
call	the original function call,
formula	the original formula,
levels	levels of the categorical regressors,
contrasts	a list with elements "count" and "zero" containing the contrasts corresponding to <code>levels</code> from the respective models,
model	the full model frame (if <code>model = TRUE</code>),
y	the response count vector (if <code>y = TRUE</code>),
x	a list with elements "count" and "zero" containing the model matrices from the respective models (if <code>x = TRUE</code>),

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

- Cameron, A. Colin and Pravin K. Trivedi. 1998. *Regression Analysis of Count Data*. New York: Cambridge University Press.
- Cameron, A. Colin and Pravin K. Trivedi. 2005. *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.
- Lambert, Diane. 1992. "Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing." *Technometrics*.V34(1):1-14

See Also

[zeroinfl.control](#), [glm](#), [glm.fit](#), [glm.nb](#), [hurdle](#)

Examples

```
## from Long (1997)
data("bioChemists", package = "pscl")

## without inflation
## ("art ~ ." is "art ~ fem + mar + kid5 + phd + ment")
fm_pois <- glm(art ~ ., data = bioChemists, family = poisson)
fm_qpois <- glm(art ~ ., data = bioChemists, family = quasipoisson)
fm_nb <- glm.nb(art ~ ., data = bioChemists)

## with simple inflation
## (no regressors for 0 component)
fm_zip <- zeroinfl(art ~ ., data = bioChemists)
fm_zinb <- zeroinfl(art ~ ., data = bioChemists, dist = "negbin", EM = TRUE)
```

```
## inflation with regressors (choose starting values by EM)
## ("art ~ . | ." is "art ~ fem + mar + kid5 + phd + ment | fem + mar + kid5 + phd + ment")
fm_zip2 <- zeroinfl(art ~ . | ., data = bioChemists, EM = TRUE)
fm_zinb2 <- zeroinfl(art ~ . | ., data = bioChemists, dist = "negbin", EM = TRUE)
```

zeroinfl.control *Control Parameters for Zero-inflated Count Data Regression*

Description

Various parameters that control fitting of zero-inflated regression models using `zeroinfl`.

Usage

```
zeroinfl.control(method = "BFGS", maxit = 10000, trace = FALSE,
  EM = FALSE, start = NULL, ...)
```

Arguments

<code>method</code>	characters string specifying the method argument passed to <code>optim</code> .
<code>maxit</code>	integer specifying the <code>maxit</code> argument (maximal number of iterations) passed to <code>optim</code> .
<code>trace</code>	logical or integer controlling whether tracing information on the progress of the optimization should be produced (passed to <code>optim</code>).
<code>EM</code>	logical. Should starting values be estimated by the EM (expectation maximization) algorithm? See details.
<code>start</code>	an optional list with elements "count" and "zero" (and potentially "theta") containing the coefficients for the corresponding component.
<code>...</code>	arguments passed to <code>optim</code> .

Details

All parameters in `zeroinfl` are estimated by maximum likelihood using `optim` with control options set in `zeroinfl.control`. Most arguments are passed on directly to `optim`, only `trace` is also used within `zeroinfl` and `EM/start` control the choice of starting values for calling `optim`.

Starting values can be supplied, estimated by the EM (expectation maximization) algorithm, or by `glm.fit` (the default). The latter corresponds to the first iteration of the EM algorithm and initializes the unobserved state as $y > 0$, i.e., all zeros are in the perfect component and only the non-zero counts in the count component. Standard errors are derived numerically using the Hessian matrix returned by `optim`. To supply starting values, `start` should be a list with elements "count" and "zero" and potentially "theta" (for negative binomial components only) containing the starting values for the coefficients of the corresponding component of the model.

Value

A list with the arguments specified.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

See Also

[zeroinfl](#)

Examples

```
data("bioChemists", package = "pscl")

## default start values
fm1 <- zeroinfl(art ~ ., data = bioChemists)

## use EM algorithm for start values
fm2 <- zeroinfl(art ~ ., data = bioChemists, EM = TRUE)

## user-supplied start values
fm3 <- zeroinfl(art ~ ., data = bioChemists,
  start = list(count = c(0.7, -0.2, 0.1, -0.2, 0, 0), zero = -1.7))
```

Index

- *Topic **classes**
 - idealToMCMC, 24
 - predict.ideal, 39
 - summary.ideal, 55
 - summary.rollcall, 57
 - *Topic **datagen**
 - constrain.items, 8
 - constrain.legis, 10
 - *Topic **datasets**
 - AustralianElections, 1
 - bioChemists, 5
 - ca2006, 5
 - partycodes, 30
 - prussian, 45
 - readKH, 46
 - s109, 51
 - sc9497, 52
 - state.info, 54
 - unionDensity, 60
 - *Topic **distribution**
 - betaHPD, 3
 - igamma, 25
 - *Topic **hplot**
 - plot.ideal, 30
 - plot.predict.ideal, 32
 - plot.seatsVotes, 33
 - tracex, 58
 - *Topic **manip**
 - computeMargins, 7
 - convertCodes, 12
 - dropRollCall, 13
 - dropUnanimous, 15
 - rollcall, 48
 - vectorRepresentation, 61
 - *Topic **methods**
 - predprob.ideal, 44
 - *Topic **misc**
 - seatsVotes, 53
 - *Topic **models**
 - extractRollCallObject, 16
 - ideal, 21
 - postProcess, 35
 - predprob, 42
 - predprob.glm, 43
 - predprob.ideal, 44
 - vuong, 62
 - *Topic **print**
 - ntable, 28
 - *Topic **regression**
 - hurdle, 17
 - hurdle.control, 19
 - odTest, 29
 - predict.hurdle, 37
 - predict.zeroinfl, 41
 - predprob, 42
 - predprob.glm, 43
 - zeroinfl, 63
 - zeroinfl.control, 66
 - *Topic **utilities**
 - vectorRepresentation, 61
- AIC, 38, 42
- alist, 7, 8, 10, 13, 14, 21, 57
- AustralianElections, 1
- base, 26
- betaHPD, 3
- binomial, 23, 43
- bioChemists, 5
- ca2006, 5
- call, 23, 53
- coef, 18, 38, 41, 64
- coef.hurdle (*predict.hurdle*), 37
- coef.zeroinfl (*predict.zeroinfl*), 41
- computeMargins, 7
- constrain.items, 8, 11, 22, 24
- constrain.legis, 8, 9, 10, 10, 22–24
- convertCodes, 7, 12
- data.frame, 47, 49
- Date, 2, 47
- dbeta, 4
- densigamma (*igamma*), 25
- density, 34
- dgamma, 27

- digamma, 26
- dimnames, 50, 59
- dropRollCall, 7–11, 13, 15, 16, 21, 57, 61
- dropUnanimous, 14, 15
- environment, 14
- eval, 14, 40
- expression, 13, 14, 55
- extractRollCallObject, 16
- factor, 28, 45
- fitted, 18, 38, 42, 64
- fitted.hurdle (*predict.hurdle*), 37
- fitted.zeroinfl
(*predict.zeroinfl*), 41
- formula, 29
- gamma, 26, 27
- glm, 19, 29, 65
- glm.fit, 17, 19, 20, 64–66
- glm.nb, 19, 29, 43, 65
- glms, 23
- hurdle, 17, 19, 20, 38, 65
- hurdle.control, 17, 18, 19, 19, 20
- ideal, 8, 9, 11, 14–16, 21, 24, 25, 31–33, 35,
36, 39, 40, 44, 50, 55, 56, 58, 59
- idealToMCMC, 24, 24
- igamma, 25
- igammaHDR (*igamma*), 25
- link, 23
- list, 7, 8, 10, 13, 21, 23, 49, 50, 55–58
- lm, 14
- logical, 14, 21, 22, 31
- logLik, 18, 29, 38, 42, 64
- logLik.hurdle (*predict.hurdle*), 37
- logLik.zeroinfl
(*predict.zeroinfl*), 41
- matched call, 58
- matrix, 12, 15, 23, 44, 49, 50, 55, 61
- mcmc, 25
- MCMCirt1d, 24
- MCMCirtKd, 24
- model.frame, 17, 63
- model.matrix, 18, 38, 42, 64
- model.matrix.hurdle
(*predict.hurdle*), 37
- model.matrix.zeroinfl
(*predict.zeroinfl*), 41
- ntable, 28
- numeric, 23
- odTest, 29
- optim, 17, 18, 20, 64, 66
- optimize, 4
- pairs, 31
- partial matches, 59
- partycodes, 30
- pbeta, 4
- pgamma, 27
- pigamma (*igamma*), 25
- plot.ideal, 24, 30
- plot.predict.ideal, 32, 40
- plot.seatsVotes, 33, 54
- plot1d, 31
- plot1d (*plot.ideal*), 30
- plot2d, 31
- plot2d (*plot.ideal*), 30
- pmatch, 59
- poisson, 43
- postProcess, 22, 24, 35
- predict, 18, 36, 38, 42, 64
- predict.glm, 44
- predict.hurdle, 18, 37
- predict.ideal, 7, 24, 33, 39, 44
- predict.zeroinfl, 41, 64
- predprob, 42, 44
- predprob.glm, 43, 43
- predprob.hurdle (*predict.hurdle*),
37
- predprob.ideal, 44
- predprob.zeroinfl, 43
- predprob.zeroinfl
(*predict.zeroinfl*), 41
- print, 18, 38, 41, 64
- print.predict.ideal
(*predict.ideal*), 39
- print.summary.hurdle
(*predict.hurdle*), 37
- print.summary.rollcall
(*summary.rollcall*), 57
- print.summary.zeroinfl
(*predict.zeroinfl*), 41
- print.zeroinfl (*zeroinfl*), 63
- prop.table, 28
- prussian, 45
- qbeta, 4
- qgamma, 27
- qigamma (*igamma*), 25
- quantiles, 59
- readKH, 30, 46, 50, 54

residuals, 18, 38, 42, 64
residuals.hurdle
 (predict.hurdle), 37
residuals.zeroinfl
 (predict.zeroinfl), 41
rigamma (igamma), 25
rollcall, 7–16, 21, 24, 33, 40, 47, 48, 48,
 51, 57, 58, 61, 62
rownames, 47
rug, 34

s109, 51
sc9497, 52
seatsVotes, 34, 53
state, 55
state.abb, 54
state.info, 54
summary, 18, 38, 41, 64
summary.hurdle (predict.hurdle),
 37
summary.ideal, 7, 24, 40, 55
summary.rollcall, 14, 15, 47, 50, 57
summary.zeroinfl
 (predict.zeroinfl), 41

table, 28
terms, 18, 38, 42, 64
terms.hurdle (predict.hurdle), 37
terms.zeroinfl
 (predict.zeroinfl), 41
tracex, 24, 32, 58

unevaluated, 55
unionDensity, 60
uniroot, 4, 26, 27

vcov, 18, 38, 41, 64
vcov.hurdle (predict.hurdle), 37
vcov.zeroinfl (predict.zeroinfl),
 41
vectorRepresentation, 61
vuong, 62

zeroinfl, 19, 41, 42, 63, 66, 67
zeroinfl.control, 63–65, 66, 66