Package 'BMS'

August 3, 2022

Type Package

Title Bayesian Model Averaging Library

Version 0.3.5

Date 2022-07-28

Author Martin Feldkircher and Stefan Zeugner and Paul Hofmarcher

Maintainer Stefan Zeugner <stefan.zeugner@ec.europa.eu>

Depends methods, stats, graphics, R (>= 2.10)

Description Bayesian Model Averaging for linear models with a wide choice of (customizable) priors. Built-in priors include coefficient priors (fixed, flexible and hyper-g priors), 5 kinds of model priors, moreover model sampling by enumeration or various MCMC approaches. Post-processing functions allow for inferring posterior inclusion and model probabilities, various moments, coefficient and predictive densities. Plotting functions available for posterior model size, MCMC convergence, predictive and coefficient densities, best models representation, BMA comparison.

License LGPL-3

URL http://bms.zeugner.eu

RoxygenNote 7.2.1

LazyData true

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

R topics documented:

`as.zlm` *Extract a Model from a bma Object*

Description

Extracts a model out of a `bma` object's saved models and converts it to a `zlm` linear model

Usage

```
as.zlm(bmao, model = 1)
```

Arguments

bmao	A bma object, e.g. resulting from a call to bms
model	The model index, in one of the following forms: An integer, denoting the rank of the model (1 for best, 2 for second-best, ...) A numeric or logical vector of length K describing which covariates are contained in the model A hexcode character describing which covariates are contained in the model

Details

A bma object stores several 'best' models it encounters (cf. argument nmodel in bms). as.zlm extracts a single model and converts it to an object of class zlm, which represents a linear model estimated under Zellner's g prior.
The utility model.frame allows to transfrom a zlm model into an OLS model of class lm.

Value

a list of class zlm

Author(s)

Stefan Zeugner

See Also

bms for creating bma objects, zlm for creating zlm objects, pmp.bma for displaying the topmodels in a bma object

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)

mm=bms(datafls[,1:6],mcmc="enumeration") # do a small BMA chain
topmodels.bma(mm)[,1:5] #display the best 5 models

m2a=as.zlm(mm,4) #extract the fourth best model
summary(m2a)

# Bayesian Model Selection:
# transform the best model into an OLS model:
lm(model.frame(as.zlm(mm)))

# extract the model only containing the 5th regressor
m2b=as.zlm(mm,c(0,0,0,0,1))

# extract the model only containing the 5th regressor in hexcode
print(bin2hex(c(0,0,0,0,1)))
m2c=as.zlm(mm,"01")
```

`beta.draws.bma` *Coefficients of the Best Models*

Description

Returns a matrix whose columns are the (expected value or standard deviations of) coefficients for the best models in a bma object.

Usage

```
beta.draws.bma(bmao, stdev = FALSE)
```

Arguments

`bmao`	a 'bma' object (as e.g. resulting from `bms`)
`stdev`	if `stdev=FALSE` then `beta.draws.bma` returns the (conditional) posterior expected values of the coefficients (i.e. 'Bayesian coefficients'). If `stdev=TRUE` it returns their posterior standard deviations.

Value

Each column presents the coefficients for the model indicated by its column name. The zero coefficients are the excluded covariates per model. Note that the coefficients returned are only those of the best (100) models encountered by the `bma` object (cf. argument `nmodels` of `bms`).

For aggregate coefficients please refer to `coef.bma`.

Note

Note that the elements of `beta.draws.bma(bmao)` correspond to `bmao$topmod$betas()`

See Also

`bms` for creating bms objects, `coef.bma` for aggregate coefficients

Check `http://bms.zeugner.eu` for additional help.

Examples

```
#sample a bma object:
data(datafls)
mm=bms(datafls,burn=500,iter=5000,nmodel=20)

#coefficients for all
beta.draws.bma(mm)
```

```
#standard deviations for the fourth- to eight best models
beta.draws.bma(mm[4:8],TRUE);
```

---

bma-class *Class "bma"*

Description

A list holding results from a BMA iteration chain

Objects from the Class

Objects can be created via calls to bms, but indirectly also via c.bma
A bma object is a list whose elements hold information on input and output for a Bayesian Model Averaging iteration chain, such as from a call to bms:

Author(s)

Martin Feldkircher and Stefan Zeugner

References

http://bms.zeugner.eu

See Also

bms for creating bma objects,
or topmod for the topmod object

Examples

```
data(datafls)
mm=bms(datafls)
#show posterior model size
print(mm$info$msize/mm$info$cumsumweights)
#is the same number as in
summary(mm)
```

bms *Bayesian Model Sampling and Averaging*

Description

Given data and prior information, this function samples all possible model combinations via MC3 or enumeration and returns aggregate results.

Usage

```
bms(
  X.data,
  burn = 1000,
  iter = NA,
  nmodel = 500,
  mcmc = "bd",
  g = "UIP",
  mprior = "random",
  mprior.size = NA,
  user.int = TRUE,
  start.value = NA,
  g.stats = TRUE,
  logfile = FALSE,
  logstep = 10000,
  force.full.ols = FALSE,
  fixed.reg = numeric(0)
)
```

Arguments

X.data	a data frame or a matrix, with the dependent variable in the first column, followed by the covariates (alternatively, X.data can also be provided as a formula). Note that bms automatically estimates a constant, therefore including constant terms is not necessary.
burn	The (positive integer) number of burn-in draws for the MC3 sampler, defaults to 1000. (Not taken into account if mcmc="enumerate")
iter	If mcmc is set to an MC3 sampler, then this is the number of iteration draws to be sampled (ex burn-ins), default 3000 draws. If mcmc="enumerate", then iter is the number of models to be sampled, starting from 0 (defaults to $2^K - 1$) - cf. start.value.
nmodel	the number of best models for which information is stored (default 500). Best models are used for convergence analysis between likelihoods and MCMC frequencies, as well as likelihood-based inference. Note that a very high value for nmodel slows down the sampler significantly. Set nmodel=0 to speed up sampling (if best model information is not needed).

`mcmc` a character denoting the model sampler to be used.
The MC3 sampler `mcmc="bd"` corresponds to a birth/death MCMC algogrithm. `mcmc="rev.jump"` enacts a reversible jump algorithm adding a "swap" step to the birth / death steps from "bd".
Alternatively, the entire model space may be fully enumerated by setting `mcmc="enumerate"` which will iterate all possible regressor combinations (Note: consider that this means 2^K iterations, where K is the number of covariates.)
Default is full enumeration (`mcmc="enumerate"`) with less then 15 covariates, and the birth-death MC3 sampler (`mcmc="bd"`) with 15 covariates or more. Cf. section 'Details' for more options.

`g` the hyperparameter on Zellner's g-prior for the regression coefficients.
`g="UIP"` corresponds to $g = N$, the number of observations (default);
`g="BRIC"` corresponds to the benchmark prior suggested by Fernandez, Ley and Steel (2001), i.e $g = max(N, K^2)$, where K is the total number of covariates;
`g="RIC"` sets $g = K^2$ and conforms to the risk inflation criterion by George and Foster (1994)
`g="HQ"` sets $g = log(N)^3$ and asymptotically mimics the Hannan-Quinn criterion with $C_{HQ} = 3$ (cf. Fernandez, Ley and Steel, 2001, p.395)
`g="EBL"` estimates a local empirical Bayes g-parameter (as in Liang et al. (2008));
`g="hyper"` takes the 'hyper-g' prior distribution (as in Liang et al., 2008) with the default hyper-parameter a set such that the prior expected shrinkage factor conforms to 'UIP';
This hyperparameter a can be adjusted (between $2 < a <= 4$) by setting `g="hyper=2.9"`, for instance.
Alternatively, `g="hyper=UIP"` sets the prior expected value of the shrinkage factor equal to that of UIP (default), `g="hyper=BRIC"` sets it according to BRIC
cf section 'Details' fro more on the hyper-g prior

`mprior` a character denoting the model prior choice, defaulting to "random":
`mprior="fixed"` denotes fixed common prior inclusion probabilities for each regressor as e.g. in Sala-i-Martin, Doppelhofer, and Miller(2004) - for their fine-tuning, cf. `mprior.size`. Preferable to `mcmc="random"` if strong prior information on model size exists;
`mprior="random"` (default) triggers the 'random theta' prior by Ley and Steel (2008), who suggest a binomial-beta hyperprior on the a priori inclusion probability;
`mprior="uniform"` employs the uniform model prior;
`mprior="customk"` allows for custom model size priors (cf. `mprior.size`);
`mprior="pip"` allows for custom prior inclusion probabilities (cf. `mprior.size`);
Note that the prior on models with more than N-3 regressors is automatically zero: these models will not be sampled.

`mprior.size` if `mprior` is "fixed" or "random", `mprior.size` is a scalar that denotes the prior expected value of the model size prior (default K/2).
If `mprior="customk"` then a custom model size prior can be provided as a K+1 vector detailing the priors from model size 0 to K (e.g. rep(1,K+1) for the uniform model prior);
if `mprior="pip"`, then custom prior inclusion probabilities can be provided as a vector of size K, with elements in the interval (0,1)

`user.int` 'interactive mode': print out results to console after ending the routine and plots

a chart (default TRUE).

`start.value` specifies the starting model of the iteration chain. For instance a specific model by the corresponding column indices (e.g. starting.model=numeric(K) starts from the null model including solely a constant term) or `start.value=c(3,6)` for a starting model only including covariates 3 and 6.
If `start.model` is set to an integer (e.g. `start.model=15`) then that number of covariates (here: 15 covariates) is randomly chosen and the starting model is identified by those regressors with an OLS t-statistic>0.2.
The default value `start.value=NA` corresponds to `start.value=min(ncol(X.data),nrow(X.data)-3`
Note that `start.value=0` or `start.value=NULL` starts from the null model.
If `mcmc="enumerate"` then `start.value` is the index to start the iteration (default: 0, the null model) . Any number between 0 and $K^2 - 1$ is admissible.

`g.stats` TRUE if statistics on the shrinkage factor g/(1+g) should be collected, defaulting to TRUE (Note: set `g.stats=FALSE` for faster iteration.)

`logfile` setting `logfile=TRUE` produces a logfile named `"test.log"` in your current working directory, in order to keep track of the sampling procedure. `logfile` equal to some filepath (like `logfile="subfolder/log.txt"`) puts the logfile into that specified position. (default: `logfile=FALSE`). Note that `logfile=""` implies log printouts on the console.

`logstep` specifies at which number of posterior draws information is written to the log file; default: 10 000 iterations

`force.full.ols` default FALSE. If `force.full.ols=TRUE`, the OLS estimation part of the sampling procedure relies on slower matrix inversion, instead of streamlined routines. `force.full.ols=TRUE` can slow down sampling but may deal better with highly collinear data

`fixed.reg` indices or variable names of `X.data` that are fixed regressors to be always included in every sampled model. Note: the parameter `mprior.size` refers to prior model size including these fixed regressors.

Details

Ad `mcmc`:
Interaction sampler: adding an ".int" to an MC3 sampler (e.g. "mcmc="bd.int") provides for special treatment of interaction terms. Interaction terms will only be sampled along with their component variables: In the colnumn names of X.data, interaction terms need to be denominated by names consisting of the base terms separated by # (e.g. an interaction term of base variables `"A"`, `"B"` and `"C"` needs column name `"A#B#C"`). Then variable `"A#B#C"` will only be included in a model if all of the component variables ("A", "B", and "C") are included.

The MC3 samplers "bd", "`rev.jump`", "`bd.int`" and "`rev.jump.int`", iterate away from a starting model by adding, dropping or swapping (only in the case of rev.jump) covariates.

In an MCMC fashion, they thus randomly draw a candidate model and then move to it in case its marginal likelihood (marg.lik.) is superior to the marg.lik. of the current model.

In case the candidate's marg.lik is inferior, it is randomly accepted or rejected according to a probability formed by the ratio of candidate marg.lik over current marg.lik. Over time, the sampler should thus converge to a sensible distribution. For aggregate results based on these MC3 frequencies, the first few iterations are typically disregarded (the 'burn-ins').

Ad g and the hyper-g prior: The hyper-g prior introduced by Liang et al. (2008) puts a prior distribution on the shrinkage factor $g/(1+g)$, namely a Beta distribution $Beta(1, 1/2-1)$ that is governed by the parameter a. $a = 4$ means a uniform prior distribution of the shrinkage factor, while $a > 2$ close to 2 concentrates the prior shrinkage factor close to one.
The prior expected value is $E(g/1+g)) = 2/a$. In this sense `g="hyper=UIP"` and `g="hyper=BRIC"` set the prior expected shrinkage such that it conforms to a fixed UIP-g (eqng=N) or BRIC-g ($g = max(K^2, N)$).

Value

A list of class `bma`, that may be displayed using e.g. `summary.bma` or `coef.bma`. The list contains the following elements:

`info`	a list of aggregate statistics: `iter` is the number of iterations, `burn` the number of burn-ins. The following have to be divided by `cumsumweights` to get posterior expected values: `inccount` are the posterior inclusion probabilities, `b1mo` and `b2mo` the first and second moment of coefficients, `add.otherstats` other statistics of interest (typically the moments of the shrinkage factor), `msize` is the post. expected model size, `k.vec` the posterior model size distribution, `pos.sign` the unconditional post. probability of positive coefficients, `corr.pmp` is the correlation between the best models' MCMC frequencies and their marg. likelihoods. `timed` is the time that was needed for MCMC sampling, `cons` is the posterior expected value of the constant. `K` and `N` are the maximum number of covariates and the sample size, respectively.
`arguments`	a list of the evaluated function arguments provided to `bms` (see above)
`topmod`	a 'topmod' object containing the best drawn models. see `topmod` for more details
`start.pos`	the positions of the starting model. If bmao is a'bma' object this corresponds to covariates bmao$reg.names[bmao$start.pos]. If bmao is a chain that resulted from several starting models (cf. `c.bma`, then `start.pos` is a list detailing all of them.
`gprior.info`	a list of class `gprior-class`, detailing information on the g-prior: `gtype` corresponds to argument g above, `is.constant` is FALSE if `gtype` is either "hyper" or "EBL", `return.g.stats` corresponds to argument `g.stats` above, `shrinkage.moments` contains the first and second moments of the shrinkage factor (only if `return.g.stats==TRUE`), g details the fixed g (if `is.constant==TRUE`), `hyper.parameter` corresponds to the hyper-g parameter a as in Liang et al. (2008)
`mprior.info`	a list of class `mprior-class`, detailing information on the model prior: `origargs` lists the original arguments to `mprior` and `mprior.size` above; `mp.msize` denotes the prior mode size; `mp.Kdist` is a (K+1) vector with the prior model size distribution from 0 to K
`X.data`	data.frame or matrix: corresponds to argument `X.data` above, possibly cleaned for NAs
`reg.names`	character vector: the covariate names to be used for X.data (corresponds to `variable.names.bma`
`bms.call`	the original call to the `bms` function

Theoretical background

The models analyzed are Bayesian normal-gamma conjugate models with improper constant and variance priors akin to Fernandez, Ley and Steel (2001): A model M can be described as follows, with $\epsilon \sim N(0, \sigma^2 I)$:

$$latex$$

$$f(\beta|\sigma, M, g)\ N(0, g\sigma^2(X'X)^{-}1)$$

Moreover, the (improper) prior on the constant $f(\alpha)$ is put proportional to 1. Similarly, the variance prior $f(\sigma)$ is proportional to $1/\sigma$.

Note

There are several ways to speed-up sampling: `nmodel=10` saves only the ten best models, at most a marginal improvement. `nmodels=0` does not save the best (500) models, however then posterior convergence and likelihood-based inference are not possible. the best models, but not their coefficients, which renders the use of `image.bma` and the paramer `exact=TRUE` in functions such as `coef.bma` infeasible. `g.stats=FALSE` saves some time by not retaining the shrinkage factors for the MC3 chain (and the best models). `force.fullobject=TRUE` in contrast, slows sampling down significantly if `mcmc="enumerate"`.

Author(s)

Martin Feldkircher, Paul Hofmarcher, and Stefan Zeugner

References

http://bms.zeugner.eu: BMS package homepage with help and tutorials

Feldkircher, M. and S. Zeugner (2015): Bayesian Model Averaging Employing Fixed and Flexible Priors: The BMS Package for R, Journal of Statistical Software 68(4).

Feldkircher, M. and S. Zeugner (2009): Benchmark Priors Revisited: On Adaptive Shrinkage and the Supermodel Effect in Bayesian Model Averaging, IMF Working Paper 09/202.

Fernandez, C. E. Ley and M. Steel (2001): Benchmark priors for Bayesian model averaging. Journal of Econometrics 100(2), 381–427

Ley, E. and M. Steel (2008): On the Effect of Prior Assumptions in Bayesian Model Averaging with Applications to Growth Regressions. working paper

Liang, F., Paulo, R., Molina, G., Clyde, M. A., and Berger, J. O. (2008). Mixtures of g Priors for Bayesian Variable Selection. Journal of the American Statistical Association 103, 410-423.

Sala-i-Martin, X. and G. Doppelhofer and R.I. Miller (2004): Determinants of long-term growth: a Bayesian averaging of classical estimates (BACE) approach. American Economic Review 94(4), 813–835

See Also

`coef.bma`, `plotModelsize` and `density.bma` for some operations on the resulting 'bma' object, `c.bma` for integrating separate MC3 chains and splitting of sampling over several runs.

Check `http://bms.zeugner.eu` for additional help.

Examples

```
data(datafls)
#estimating a standard MC3 chain with 1000 burn-ins and 2000 iterations and uniform model priors
bma1 = bms(datafls,burn=1000, iter=2000, mprior="uniform")

##standard coefficients based on exact likelihoods of the 100 best models:
coef(bma1,exact=TRUE, std.coefs=TRUE)

#suppressing user-interactive output, using a customized starting value, and not saving the best
#  ...models for only 19 observations (but 41 covariates)
bma2 = bms(datafls[20:39,],burn=1000, iter=2000, nmodel=0, start.value=c(1,4,7,30),
   user.int=FALSE)
coef(bma2)

#MC3 chain with a hyper-g prior (custom coefficient a=2.1), saving only the 20 best models,
# ...and an alternative sampling procedure; putting a log entry to console every 1000th step
bma3 = bms(datafls,burn=1000, iter=5000, nmodel=20, g="hyper=2.1", mcmc="rev.jump",
    logfile="",logstep=1000)
image(bma3) #showing the coefficient signs of the 20 best models

#enumerating with 10 covariates (= 1024 models), keeping the shrinkage factors
#  ...of the best 200 models
bma4 = bms(datafls[,1:11],mcmc="enumerate",nmodel=200,g.stats=TRUE)

#using an interaction sampler for two interaction terms
dataint=datafls
dataint=cbind(datafls,datafls$LifeExp*datafls$Abslat/1000,
      datafls$Protestants*datafls$Brit-datafls$Muslim)
names(dataint)[ncol(dataint)-1]="LifeExp#Abslat"
names(dataint)[ncol(dataint)]="Protestants#Brit#Muslim"
bma5 = bms(X.data=dataint,burn=1000,iter=9000,start.value=0,mcmc="bd.int")

density(bma5,reg="English") # plot posterior density for covariate "English"

# a matrix as X.data argument
bms(matrix(rnorm(1000),100,10))

# keeping a set of fixed regressors:
bms(datafls, mprior.size=7, fixed.reg = c("PrScEnroll", "LifeExp", "GDP60"))
# Note that mprior.size=7 means prior model size of 3 fixed to 4 'uncertain' regressors
```

`c.bma` *Concatenate bma objects*

Description

Combines bma objects (resulting from `bms`). Can be used to split estimation over several machines, or combine the MCMC results obtained from different starting points.

Usage

```
## S3 method for class 'bma'
c(..., recursive = FALSE)
```

Arguments

...	At least two 'bma' objects (cf. bms)
recursive	retained for compatibility with c method

Details

Aggregates the information obtained from several chains. The result is a 'bma' object (cf. 'Values' in bms) that can be used just as a standard 'bma' object.
Note that combine_chains helps in particular to paralllelize the enumeration of the total model space: A model with K regressors has 2^K potential covariate combinations: With K large (more than 25), this can be pretty time intensive. With the bms arguments start.value and iter, sampling can be done in steps: cf. example 'enumeration' below.

See Also

bms for creating bma objects

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)

#MCMC case ############################
model1=bms(datafls,burn=1000,iter=4000,mcmc="bd",start.value=c(20,30,35))
model2=bms(datafls,burn=1500,iter=7000,mcmc="bd",start.value=c(1,10,15))

model_all=c(model1,model2)
coef(model_all)
plot(model_all)



#splitting enumeration ########################

#standard case with 12 covariates (4096 differnt combinations):
enum0=bms(datafls[,1:13],mcmc="enumerate")

# now split the task:
# enum1 does everything from model zero (the first model) to model 1999
enum1=bms(datafls[,1:13],mcmc="enumerate",start.value=0,iter=1999)

# enum2 does models from index 2000 to the index 3000 (in total 1001 models)
enum2=bms(datafls[,1:13],mcmc="enumerate",start.value=2000,iter=1000)

# enum3 does models from index 3001 to the end
```

```
enum3=bms(datafls[,1:13],mcmc="enumerate",start.value=3001)

enum_combi=c(enum1,enum2,enum3)
coef(enum_combi)
coef(enum0)
#both enum_combi and enum0 have exactly the same results
#(one difference: enum_combi has more 'top models' (1500 instead of 500))
```

`datafls` *FLS (2001) growth data*

Description

The economic growth data set from Fernandez, Ley and Steel, Journal of Applied Econometrics 2001

Usage

```
datafls
```

Format

A data frame with 53940 rows and 10 variables: A data frame with 72 observations on the following 42 variables.

`y` numeric: Economic growth 1960-1992 as from the Penn World Tables Rev 6.0

`Abslat` numeric: Absolute latitude

`Spanish` numeric: Spanish colony dummy

`French` numeric: French colony dummy

`Brit` numeric: British colony dummy

`WarDummy` numeric: War dummy

`LatAmerica` numeric: Latin America dummy

`SubSahara` numeric; Sub-Sahara dummy

`OutwarOr` numeric: Outward Orientation

`Area` numeric: Area surface

`PrScEnroll` numeric: Primary school enrolment

`LifeExp` numeric: Life expectancy

`GDP60` numeric: Initial GDP in 1960

`Mining` numeric: Fraction of GDP in mining

`EcoOrg` numeric: Degree of capitalism

`YrsOpen` numeric: Number of years having an open economy

`Age` numeric: Age

`Buddha` numeric: Fraction Buddhist

`Catholic` numeric: Fraction Catholic

`Confucian` numeric: Fraction Confucian

`EthnoL` numeric: Ethnolinguistic fractionalization

`Hindu` numeric: Fraction Hindu

`Jewish` numeric: Fraction Jewish

`Muslim` numeric: Fraction Muslim

`PrExports` numeric: Primary exports 1970

`Protestants` numeric: Fraction Protestants

`RuleofLaw` numeric: Rule of law

`Popg` numeric: Population growth

`WorkPop` numeric: workers per inhabitant

`LabForce` numeric: Size of labor force

`HighEnroll` numeric: Higher education enrolment

`PublEdupct` numeric: Public education share

`RevnCoup` numeric: Revolutions and coups

`PolRights` numeric: Political rights

`CivlLib` numeric: Civil liberties

`English` numeric: Fraction speaking English

`Foreign` numeric: Fraction speaking foreign language

`RFEXDist` numeric: Exchange rate distortions

`EquipInv` numeric: Equipment investment

`NequipInv` numeric: Non-equipment investment

`stdBMP` numeric: stand. dev. of black market premium

`BlMktPm` numeric: black market premium

Source

Fernandez, C., Ley, E., and Steel, M. F. (2001b). Model Uncertainty in Cross-Country Growth Regressions. Journal of Applied Econometrics, 16:563-576. Data set from https://www2.warwick.ac.uk/fac/sci/statistics/staff/academic/steel/steel_homepage/software. A working paper version of Fernandez, Ley and Steel (2001) is available at https://www2.warwick.ac.uk/fac/sci/statistics/staff/academic/steel/steel_homepage/software/fls3fin.pdf.

density.bma *Coefficient Marginal Posterior Densities*

Description

Calculates the mixture marginal posterior densities for the coefficients from a BMA object and plots them

Usage

```
## S3 method for class 'bma'
density(
  x,
  reg = NULL,
  addons = "lemsz",
  std.coefs = FALSE,
  n = 300,
  plot = TRUE,
  hnbsteps = 30,
  addons.lwd = 1.5,
  ...
)
```

Arguments

x	A bma object (see bms) or a zlm object.
reg	A scalar integer or character detailing which covariate's coefficient should be plotted. If reg=NULL (default), then all regressors are plotted one after the other, waiting for user interaction.
addons	character. Specifies which additional information should be added to the plot via low-level commands (see 'Details' below).
std.coefs	logical. If TRUE then the posterior density is estimated for standardized coefficients (representing the case where all variables have mean zero and standard deviation 1) - default is FALSE.
n	numeric. the number of equally spaced points at which the density is to be estimated.
plot	logical. If TRUE (default), the density is plotted; if FALSE then density.bma only returns the estimated posterior densities without plotting.
hnbsteps	even integer, default 30. The number of numerical integration steps to be used in case of a hyper-g prior (cf. argument g in bms). Increase this number to increase accuracy.
addons.lwd	scalar, default 1.5. Line width to be used for the low-level plotting commands specified by addons. Cf. argument lwd in par
...	Additional arguments for plot.default with sensible defaults

Details

The argument addons specifies what additional information should be added to the plot(s) via the low-level commands lines and legend:
"e" for the posterior expected value (EV) of coefficients conditional on inclusion (see argument exact=TRUE in coef.bma),
"s" for 2 times posterior standard deviation (SD) bounds,
"m" for the posterior median,
"b" for posterior expected values of the individual models whom the density is averaged over,
"E" for posterior EV under MCMC frequencies (see argument exact=FALSE in coef.bma),
"S" for the corresponding SD bounds (MCMC),
"p" for plotting the Posterior Inclusion Probability above the density plot,
"l" for including a legend, "z" for a zero line, "g" for adding a grid

Any combination of these letters will give the desired result. Use addons="" for not using any of these.
In case of density.zlm, only the letters e, s, l, z, and g will have an effect.

Value

The function returns a list containing objects of the class density detailing the marginal posterior densities for each coefficient provided in reg.
In case of density.zlm, simple marginal posterior coefficient densities are computed, while density.bma calculates there mixtures over models according to posterior model probabilities.
These densities contain only the density points apart from the origin. (see 'Note' below)

As long as plot=TRUE, the densities are plotted too. Note that (for density.bma) if the posterior inclusion probability of a covariate is zero, then it will not be plotted, and the returned density will be list(x=numeric(n),y=numeric(n)).

Note

The computed marginal posterior densities from density.bma are a Bayesian Model Averaging mixture of the marginal posterior densities of the individual models. The accuracy of the result therefore depends on the number of 'best' models contained in x (cf. argument nmodel in bms).

The marginal posterior density can be interpreted as 'conditional on inclusion': If the posterior inclusion probability of a variable is smaller than one, then some of its posterior density is Dirac at zero. Therefore the integral of the returned density vector adds up to the posterior inclusion probability, i.e. the probability that the coefficient is not zero.

Correspondingly, the posterior EV and SD specified by addons="es" are based on 'best' model likelihoods ('exact') and are conditional on inclusion. They correspond to the results from command coef.bma(x,exact=TRUE,condi.coef=TRUE,order.by.pip=FALSE) (cf. the example below).

The low-level commands enacted by the argument addons rely on colors of the palette: color 2 for "e" and "s", color 3 for "m", color 8 for "b", color 4 for "E" and "S". The default colors may be changed by a call to palette.

Up to BMS version 0.3.0, density.bma may only cope with built-in gpriors, not with any user-defined priors.

See Also

quantile.coef.density for extracting quantiles, coef.bma for similar concepts, bms for creating bma objects

Check http://bms.zeugner.eu for additional help.

Examples

```
 data(datafls)
 mm=bms(datafls)

 density(mm,reg="SubSahara")
 density(mm,reg=7,addons="lbz")
 density(mm,1:9)
 density(mm,reg=2,addons="zgSE",addons.lwd=2,std.coefs=TRUE)

# plot the posterior density only for the very best model
 density(mm[1],reg=1,addons="esz")


#using the calculated density for other purposes...
 dd=density(mm,reg="SubSahara")
 plot(dd)

 dd_list=density(mm,reg=1:3,plot=FALSE,n=400)
 plot(dd_list[[1]])


#Note that the shown density is only the part that is not zero
 dd=density(mm,reg="Abslat",addons="esl")
 pip_Abslat=sum(dd$y)*diff(dd$x)[1]

 #this pip and the EV conform to what is done by the follwing command
 coef(mm,exact=TRUE,condi.coef=TRUE)["Abslat",]
```

---

estimates.bma *Posterior Inclusion Probabilities and Coefficients from a 'bma' Object*

Description

Returns a matrix with aggregate covariate-specific Bayesian model Averaging: posterior inclusion probabilites (PIP), post. expected values and standard deviations of coefficients, as well as sign probabilites

Usage

```
estimates.bma(
  bmao,
  exact = FALSE,
  order.by.pip = TRUE,
  include.constant = FALSE,
  incl.possign = TRUE,
  std.coefs = FALSE,
  condi.coef = FALSE
)

## S3 method for class 'bma'
coef(
  object,
  exact = FALSE,
  order.by.pip = TRUE,
  include.constant = FALSE,
  incl.possign = TRUE,
  std.coefs = FALSE,
  condi.coef = FALSE,
  ...
)
```

Arguments

`exact`	if `exact=FALSE`, then PIPs, coefficients, etc. will be based on aggregate information from the sampling chain with posterior model distributions based on MCMC frequencies (except in case of enumeration - cf. 'Details'); if `exact=TRUE`, estimates will be based on the `nmodel` best models encountered by the sampling chain, with the posterior model distribution based on their *exact* marginal likelihoods - cf. 'Details' below.
`order.by.pip`	`order.by.pip=TRUE` orders the resulting matrix according to posterior inclusion probabilites, `order.by.pip=FALSE` ranks them according to the original data (order of the covariates as in provided in `X.data` to `bms`), default `TRUE`
`include.constant`	If `include.constant=TRUE` then the resulting matrix includes the expected value of the constant in its last row. Default `FALSE`
`incl.possign`	If `incl.possign=FALSE`, then the sign probabilites column (cf. 'Values' below) is omitted from the result. Default `TRUE`
`std.coefs`	If `std.coefs=TRUE` then the expected values and standard deviations are returned in standardized form, i.e. as if the original data all had mean zero and variance 1. If `std.coefs=FALSE` (default) then both expected values and standard deviations are returned 'as is'.
`condi.coef`	If `condi.coef=FALSE` (default) then coefficients β_i and standard deviations are unconditional posterior expected values, as in standard model averaging; if `condi.coef=FALSE` then they are given as conditional on inclusion (equivalent to β_i/PIP_i).
`object, bmao`	a 'bma' object (cf. `bms`)

... further arguments for other coef methods

Details

More on the argument exact:
In case the argument exact=TRUE, the PIPs, coefficient statistics and conditional sign probabilities are computed on the basis of the (500) best models the sampling chain encountered (cf. argument nmodel in bms). Here, the weights for Bayesian model averaging (BMA) are the posterior marginal likelihoods of these best models.
In case exact=FALSE, then these statistics are based on all accepted models (except burn-ins): If mcmc="enumerate" then this are simply all models of the traversed model space, with their marginal likelihoods providing the weights for BMA.
If, however, the bma object bmao was based on an MCMC sampler (e.g. when bms argument mcmc="bd"), then BMA statistics are computed differently: In contrast to above, the weights for BMA are MCMC frequencies, i.e. how often the respective models were encountered by the MCMC sampler. (cf. a comparison of MCMC frequencies and marginal likelihoods for the best models via the function pmp.bma).

Value

A matrix with five columns (or four if incl.possign=FALSE)

Column 'PIP' Posterior inclusion probabilities $\sum p(\gamma|i \in \gamma, Y)/sump(\gamma|Y)$

Column 'Post Mean'
posterior expected value of coefficients, unconditional $E(\beta|Y) = \sum p(\gamma|Y)E(\beta|\gamma, Y)$, where $E(\beta_i|\gamma, i \notin \gamma, Y) = 0$ if condi.coef=FALSE, or conditional on inclusion $(E(\beta|Y)/\sum p(\gamma|Y, i \in \gamma))$ if condi.coef=TRUE

Column 'Post SD'
posterior standard deviation of coefficients, unconditional or conditional on inclusion, depending on condi.coef

Column 'Cond.Pos.Sign'
The ratio of how often the coefficients' expected values were positive conditional on inclusion. (over all visited models in case exact=FALSE, over the best models in case exact=TRUE)

Column 'Idx' the original order of covariates as the were used for sampling. (if included, the constant has index 0)

See Also

bms for creating bma objects, pmp.bma for comparing MCMC frequencies and marginal likelihoods.

Check http://bms.zeugner.eu for additional help.

Examples

```
#sample, with keeping the best 200 models:
data(datafls)
mm=bms(datafls,burn=1000,iter=5000,nmodel=200)

#standard BMA PIPs and coefficients from the MCMC sampling chain, based on
```

```
#  ...how frequently the models were drawn
coef(mm)

#standardized coefficients, ordered by index
coef(mm,std.coefs=TRUE,order.by.pip=FALSE)

#coefficients conditional on inclusion:
coef(mm,condi.coef=TRUE)

#same as
ests=coef(mm,condi.coef=FALSE)
ests[,2]/ests[,1]

#PIPs, coefficients, and signs based on the best 200 models
estimates.bma(mm,exact=TRUE)

#... and based on the 50 best models
coef(mm[1:50],exact=TRUE)
```

`f21hyper` *Gaussian Hypergeometric Function F(a,b,c,z)*

Description

Computes the value of a Gaussian hypergeometric function $F(a, b, c, z)$ for $-1 \leq z \leq 1$ and $a, b, c \geq 0$

Usage

```
f21hyper(a, b, c, z)
```

Arguments

a	The parameter a of the Gaussian hypergeometric function, must be a positive scalar here
b	The parameter b of the Gaussian hypergeometric function, must be a positive scalar here
c	The parameter c of the Gaussian hypergeometric function, must be a positive scalar here
z	The parameter z of the Gaussian hypergeometric function, must be between -1 and 1 here

Details

The function `f21hyper` complements the analysis of the 'hyper-g prior' introduced by Liang et al. (2008).
For parameter values, compare cf. https://en.wikipedia.org/wiki/Hypergeometric_function.

Value

The value of the Gaussian hypergeometric function $F(a, b, c, z)$

Note

This function is a simple wrapper function of sped-up code that is intended for sporadic application by the user; it is neither efficient nor general; for a more general version cf. the package 'hypergeo'

References

Liang F., Paulo R., Molina G., Clyde M., Berger J.(2008): Mixtures of g-priors for Bayesian variable selection. J. Am. Statist. Assoc. 103, p. 410-423

https://en.wikipedia.org/wiki/Hypergeometric_function

See Also

package hypergeo for a more proficient implementation.

Check http://bms.zeugner.eu for additional help.

Examples

```
f21hyper(30,1,20,.8) #returns about 165.8197

f21hyper(30,10,20,0) #returns one

f21hyper(10,15,20,-0.1) # returns about 0.4872972
```

---

fullmodel.ssq *OLS Statistics for the Full Model Including All Potential Covariates*

Description

A utility function for reference: Returns a list with R2 and sum of squares for the OLS model encompassing all potential covariates that are included in a bma object.

Usage

```
fullmodel.ssq(yX.data)
```

Arguments

yX.data	a bma object (cf. bms) - alternatively a data.frame or matrix whose first column is the dependent variable

Value

Returns a list with some basic OLS statistics

R2	The R-squared of the full model
ymy	The sum of squares of residuals of the full model
ypy	The explained sum of squares of the full model
yty	The sum of squares of the (demeaned) dependent variable
Fstat	The F-statistic of the full model

Note

This function is just for quick comparison; for proper OLS estimation consider lm

See Also

bms for creating bma objects, lm for OLS estimation

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
mm=bms(datafls)

fullmodel.ssq(mm)

#equivalent:
fullmodel.ssq(datafls)
```

---

gdensity *Posterior Density of the Shrinkage Factor*

Description

Calculates the mixture marginal posterior density for the shrinkage factor (g/(1+g)) from a BMA object under the hyper-g prior and plots it

Usage

```
gdensity(x, n = 512, plot = TRUE, addons = "zles", addons.lwd = 1.5, ...)
```

Arguments

x	A bma object (see bms).
n	The integer number of equally spaced points at which the density is to be estimated. see 'Details' below
plot	logical. If TRUE (default), the density is plotted; if FALSE then gdensity only returns the estimated posterior density without plotting.
addons	character, defaulting to "zles". Specifies which additional information should be added to the plot via low-level commands (see 'Details' below).
addons.lwd	scalar, default 1.5. Line width to be used for the low-level plotting commands specified by addons. Cf. argument lwd in par
...	Additional arguments for plot.default with sensible defaults

Details

The function gdensity estimates and plots the posterior density for the shrinkage factor $g/(1+g)$
This is evidently only possible if the shrinkage factor if not fixed, i.e. if the bma object x was estimated with a hyper-g prior - cf. argument g in bms
The density is based only on the best models retained in the bma object x, cf. argument nmodel in bms
A note on argument n: The points at which the density is estimated start at $max(0, E - 5 * SD)$, where E and SD are the expected value and standard deviation of the shrinkage factor, respectively. For plotting the entire domain $(0, 1)$ use xlim=c(0,1) as an argument for gdensity.

The argument addons specifies what additional information should be added to the plot(s) via the low-level commands lines and legend:
"e" for the posterior expected value (EV) of the shrinkage factor,
"s" for 2 times posterior standard deviation (SD) bounds,
"m" for the posterior median,
"f" for posterior expected values of the individual models whom the density is averaged over,
"z" for a zero line, "l" for including a legend
The following two are only possible if the bma object collected statistics on shrinkage, cf. argument g.stats in bms "E" for posterior expected value under MCMC frequencies (see argument exact in coef.bma),
"S" for the corresponding 2 times standard deviation bounds (MCMC),

Any combination of these letters will give the desired result. Use addons="" for not using any of these.

Value

gdensity returns an object of the class density detailing the posterior mixture density of the shrinkage factor.

Note

The computed marginal posterior density is a Bayesian Model Averaging mixture of the marginal posterior densities of the shrinkage factor under individual models. The accuracy of the result therefore depends on the number of 'best' models contained in x (cf. argument nmodel in bms).

Correspondingly, the posterior EV and SD specified by addons="es" are based on 'best' model likelihoods ('exact') and are conditional on inclusion.

The low-level commands enacted by the argument addons rely on colors of the palette: color 2 for "e" and "s", color 3 for "m", color 8 for "f", color 4 for "E" and "S". The default colors may be changed by a call to palette.

See Also

density.bma for computing coefficient densities, bms for creating bma objects, density for the general method

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
mm=bms(datafls,g="hyper=UIP")

gdensity(mm) # default plotting

# the grey bars represent expected shrinkage factors of the individual models
gdensity(mm,addons="lzfes")

# #plotting the median 'm' and the posterior mean and bounds based on MCMC results:
gdensity(mm,addons="zSEm",addons.lwd=2)

# plot the posterior shrinkage density only for the very best model
gdensity(mm[1],addons="esz")


#using the calculated density for other purposes...
dd=gdensity(mm,plot=FALSE)
plot(dd)
```

---

gprior-class *Class "gprior"*

Description

An object pertaining to a coefficient prior

Objects from the Class

A gprior object holds descriptions and subfunctions pertaining to coefficient priors. Functions such as bms or zlm rely on this class to 'convert' the output of OLS results into posterior expressions for a Bayesian Linear Model. Post-processing functions such as density.bma also resort to gprior objects.

There are currently three coefficient prior structures built into the BMS package, generated by the following functions (cf. Feldkircher and Zeugner, 2009) :
`gprior.constg.init`: creates a Zellner's g-prior object with constant g.
`gprior.eblocal.init`: creates an Empricial Bayes Zellner's g-prior.
`gprior.hyperg.init`: creates a hyper g-prior with a Beta-prior on the shrinkage parameter.
The following describes the necessary slots

Author(s)

Martin Feldkircher and Stefan Zeugner

References

Feldkircher, M. and S. Zeugner (2009): Benchmark Priors Revisited: On Adaptive Shrinkage and the Supermodel Effect in Bayesian Model Averaging, IMF Working Paper 09/202.

See Also

`bms` and `zlm` for creating `bma` or `zlm` objects.
Check the appendix of `vignette(BMS)` for a more detailed description of built-in priors.
Check `http://bms.zeugner.eu/custompriors.php` for examples.

Examples

```
data(datafls)
mm1=bms(datafls[,1:10], g="EBL")
gg=mm1$gprior.info # is the g-prior object, augmented with some posterior statistics

mm2=bms(datafls[,1:10], g=gg) #produces the same result

mm3=bms(datafls[,1:10], g=BMS:::.gprior.eblocal.init)
#this passes BMS's internal Empirical Bayes g-prior object as the coefficient prior
# - any other obejct might be used as well
```

---

hex2bin *Converting Binary Code to and from Hexadecimal Code*

Description

A simple-to-use function for converting a logical ('binary') vector into hex code and reverse.

Usage

```
hex2bin(hexcode)

bin2hex(binvec)
```

Arguments

hexcode	a single-element character denoting an integer in hexcode (admissible character: 0 to 9, ato f)
binvec	a logical vector (alternatively a vector coercible into logical)

Details

The argument is an integer in binary form (such as "101"), provided as a logical (`c(T,F,T)`) or numeric vector (`c(1,0,1)`).
`bin2hex` then returns a character denoting this number in hexcode (in this case "5").

The function `hex2bin` does the reverse operation, e.g. `hex2bin("5")` gives (`c(1,0,1)`).

Value

`bin2hex` returns a single element character; `hex2bin` returns a numeric vector equivalent to a logical vector

See Also

`hex2bin` for converting hexcode into binary vectors, `format.hexmode` for a related R function.

Check `http://bms.zeugner.eu` for additional help.

Examples

```
bin2hex(c(TRUE,FALSE,TRUE,FALSE,TRUE,TRUE))
bin2hex(c(1,0,1,0,1,1))
hex2bin("b8a")
bin2hex(hex2bin("b8a"))
```

---

image.bma *Plot Signs of Best Models*

Description

Plots a grid with signs and inclusion of coefficients vs. posterior model probabilities for the best models in a 'bma' object:

Usage

```
## S3 method for class 'bma'
image(
  x,
  yprop2pip = FALSE,
  order.by.pip = TRUE,
  do.par = TRUE,
  do.grid = TRUE,
```

```
  do.axis = TRUE,
  cex.axis = 1,
  ...
)
```

Arguments

x	a list of class bma (cf. bms for further details)
yprop2pip	if yprop2pip=TRUE then the grid lines on the vertical axis are scaled according to the coefficients' inclusion probabilites. If yprop2pip=FALSE (default) then the grid lines on the vertical axis are equidistant.
order.by.pip	with order.by.pip=TRUE (default), coefficients are sorted according to their posterior inclusion probabilites along the vertical axis. If order.by.pip=FALSE they are ordered as they were provided to bms.
do.par	Defaults to do.par=TRUE, which adjusts par()$mar for optimal positioning. Set do.par=FALSE for customizing par yourself.
do.grid	do.grid=TRUE (default) plots grid lines among the chart's boxes, akin to the low level command grid. do.grid=FALSE omits the grid lines.
do.axis	do.axis=TRUE (default) plots axis tick marks and labels (cf. axis). do.axis=FALSE omits them.
cex.axis	font size for the axes (cf. axis), defaults to 1
...	Parameters to be passed on to image.default.

Details

Under default settings, blue corresponds to positive sign, red to a negative sign, white to non-inclusion.

See Also

coef.bma for the coefficients in matrix form, bms for creating 'bma' objects.

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)

model=bms(datafls,nmodel=200)

#plot all models
image(model,order.by.pip=FALSE)
image(model,order.by.pip=TRUE,cex.axis=.8)

#plot best 7 models, with other colors
image(model[1:7],yprop2pip=TRUE,col=c("black","lightgrey"))
```

`info.bma` *Summary Statistics for a 'bma' Object*

Description

Returns a vector with summary statistics for a 'bma' object

Usage

```
info.bma(bmao)

## S3 method for class 'bma'
summary(object, ...)
```

Arguments

`bmao`	same as `object`
`object`	a list/object of class 'bma' that typically results from the function `bms` (see `bms` for details)
`...`	further arguments passed to or from other methods

Details

`info.bma` is equivalent to `summary.bma`, its argument `bmao` conforms to the argument `object`

Value

A character vector summarizing the results of a call to `bms`

`Mean no. of Regressors`
: the posterior mean of model size

`Draws`
: the number of iterations (ex burn-ins)

`Burnins`
: the number of burn-in iterations

`Time`
: the time spent on iterating through the model space

`No. of models visited`
: the number of times a model was accepted (including burn-ins)

`Modelspace`
: the total model space 2^K

`list(list("2^K"))`
: the total model space 2^K

`Percentage visited`
: `No. of models visited/Modelspace*100`

`Percentage Topmodels`
: number of times the best models were drawn in percent of `Draws`

`Corr. PMP`
: the correlation between the MCMC frequencies of the best models (the number of times they were drawn) and their marginal likelihoods.

No. Obs.	Number of observations
Model Prior	a character conforming to the argument mprior of bms, and the expected prior model size
g-prior	a character corresponding to argument g of function bms
Shrinkage-Stats	Posterior expected value und standard deviation (if applicable) of the shrinkage factor. Only included if argument g.stats of function bms was set to TRUE

Note

All of the above statistics can also be directly extracted from the bma object (bmao). Therefore summary.bma only returns a character vector.

See Also

bms and c.bma for functions creating bma objects, print.bma makes use of summary.bma.

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)

m_fixed=bms(datafls,burn=1000,iter=3000,user.int=FALSE, )
summary(m_fixed)

m_ebl=bms(datafls,burn=1000,iter=3000,user.int=FALSE, g="EBL",g.stats=TRUE)
info.bma(m_ebl)
```

---

is.bma *Tests for a 'bma' Object*

Description

tests for objects of class "bma"

Usage

```
is.bma(bmao)
```

Arguments

bmao	a 'bma' object: see 'value'

Value

Returns TRUE if bmao is of class 'bma', FALSE otherwise.

See Also

'Output' in bms for the structure of a 'bma' object

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
mm=bms(datafls,burn=1000, iter=4000)
is.bma(mm)
```

---

lps.bma *Log Predictive Score*

Description

Computes the Log Predictive Score to evaluate a forecast based on a bma object

Usage

```
lps.bma(object, realized.y, newdata = NULL)
```

Arguments

object	an object of class pred.density, or class bma (cf. bms), or class zlm
realized.y	a vector with realized values of the dependent variables to be plotted in addition to the predictive density, must have its length conforming to newdata
newdata	Needs to be provided if object is not of class pred.density: a data.frame, matrix or vector containing variables with which to predict.

Details

The log predictive score is an indicator for the likelihood of several forecasts.
It is defined as minus the arithmethic mean of the logarithms of the point densities for realized.y given newdata.
Note that in most cases is more efficient to first compute the predictive density object via a call to pred.density and only then pass the result on to lps.bma.

Value

A scalar denoting the log predictive score

See Also

pred.density for constructing predictive densities, bms for creating bma objects, density.bma for plotting coefficient densities

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
mm=bms(datafls,user.int=FALSE,nmodel=100)

#LPS for actual values under the used data (static forecast)
lps.bma(mm, realized.y=datafls[,1] , newdata=datafls[,-1])

#the same result via predicitve.density
pd=pred.density(mm, newdata=datafls[,-1])
lps.bma(pd,realized.y=datafls[,1])

# similarly for a linear model (not BMA)
zz = zlm(datafls)
lps.bma(zz, realized.y=datafls[,1] , newdata=datafls[,-1])
```

---

mprior-class *Class "mprior"*

Description

An object pertaining to a BMA model prior

Objects from the Class

An `mprior` object holds descriptions and subfunctions pertaining to model priors. The BMA functions `bms` and post-processing functions rely on this class.
There are currently five model prior structures built into the BMS package, generated by the following functions (cf. the appendix of `vignette(BMS)`):
`mprior.uniform.init`: creates a uniform model prior object.
`mprior.fixedt.init`: creates the popular binomial model prior object with common inclusion probabilities.
`mprior.randomt.init`: creates a beta-binomial model prior object.
`mprior.pip.init`: creates a binomial model prior object that allows for defining individual prior inclusion probabilities.
`mprior.customk.init`: creates a model prior object that allows for defining a custom prior for each model parameter size.
The following describes the necessary slots:

Author(s)

Martin Feldkircher and Stefan Zeugner

See Also

`bms` for creating bma objects.
Check the appendix of `vignette(BMS)` for a more detailed description of built-in priors.
Check `http://bms.zeugner.eu/custompriors.php` for examples.

plot.bma *Plot Posterior Model Size and Model Probabilities*

Description

Produces a combined plot: upper row shows prior and posterior model size distribution, lower row shows posterior model probabilities for the best models

Usage

```
## S3 method for class 'bma'
plot(x, ...)
```

Arguments

x	an object of class 'bma'
...	additional arguments for matplot

Value

combines the plotting functions plotModelsize and plotConv

Note

The upper plot shows the prior and posterior distribution of model sizes (plotModelsize).
The lower plot is an indicator of how well the bma object has converged (plotConv). and Paul

See Also

plotModelsize and plotConv

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
mm=bms(datafls,user.int=FALSE)

plot(mm)
```

plotComp *Compare Two or More bma Objects*

Description

Plots a comparison of posterior inclusion probabilites, coefficients or their standard deviation between various bma objects

Usage

```
plotComp(
  ...,
  varNr = NULL,
  comp = "PIP",
  exact = FALSE,
  include.legend = TRUE,
  add.grid = TRUE,
  do.par = TRUE,
  cex.xaxis = 0.8
)
```

Arguments

`...`	one or more objects of class 'bma' to be compared. `plotComp` passes on any other parameters in `...{}` to `matplot`.
`varNr`	optionally, covariate indices to be included in the plot, can be either integer vector or character vector - see examples
`comp`	a character denoting what should be compared: `comp="PIP"` (default) for posterior inclusion probabilities, `comp="Post Mean"` for coefficients, `comp="Post SD"` for their standard deviations, `comp="Std Mean"` or standardized coefficients, or `comp="Std SD"` for standardized standard deviations
`exact`	if `FALSE`, the statistics to be compared are based on aggregate bma statistics, if `TRUE`, they are based solely on the best models retained in the bma objects
`include.legend`	whether to include a default legend in the plot (custom legends can be added with the command `legend`)
`add.grid`	whether to add a `grid` to the plot
`do.par`	whether to adjust `par("mar")` in order to fit in the tick labels on the x-axis
`cex.xaxis`	font size scaling parameter for the x-axis - cf. argument `cex.axis` in `par`

See Also

`coef.bma` for the underlying function

Check `http://bms.zeugner.eu` for additional help.

Examples

```
## sample two simple bma objects
data(datafls)
mm1=bms(datafls[,1:15])
mm2=bms(datafls[,1:15])

#compare PIPs
plotComp(mm1,mm2)

#compare standardized coefficeitns
plotComp(mm1,mm2,comp="Std Mean")

#...based on the lieklihoods of best models
plotComp(mm1,mm2,comp="Std Mean",exact=TRUE)

#plot only PIPs for first four covariates
plotComp(mm1,mm2,varNr=1:4, col=c("black","red"))

#plot only coefficients for covariates 'GDP60 ' and 'LifeExp'
plotComp(mm1,mm2,varNr=c("GDP60", "LifeExp"),comp="Post Mean")
```

`plotConv` *Plot Convergence of BMA Sampler*

Description

Plots the posterior model probabilites based on 1) marginal likelihoods and 2) MCMC frequencies for the best models in a 'bma' object and details the sampler's convergence by their correlation

Usage

```
plotConv(bmao, include.legend = TRUE, add.grid = TRUE, ...)
```

Arguments

`bmao`	an object of class 'bma' - see `bms`
`include.legend`	whether to include a `legend` in the plot
`add.grid`	whether to include a `grid` in the plot
`...`	other parameters for `matplot`

Details

A call to bms with a MCMC sampler (e.g. `bms(datafls,mcmc="bd",nmodel=100)` uses a Metropolis-Hastings algorithm to sample through the model space: the frequency of how often models are drawn converges to the distribution of their posterior marginal likelihoods.
While sampling, each 'bma' object stores the best models encountered by its sampling chain with their marginal likelihood and their MCMC frequencies.
`plotConv` compares the MCMC frequencies to marginal likelihoods, and thus visualizes how well the sampler has converged.

Note

`plotConv` is also used by `plot.bma`

See Also

`pmp.bma` for posterior model probabilites based on the two concepts, `bms` for creating objects of class 'bma'

Check `http://bms.zeugner.eu` for additional help.

Examples

```
data(datafls)
mm=bms(datafls[,1:12],user.int=FALSE)

plotConv(mm)

#is similar to
matplot(pmp.bma(mm),type="l")
```

---

plotModelsize *Plot Model Size Distribution*

Description

Plots posterior and prior model size distribution

Usage

```
plotModelsize(
  bmao,
  exact = FALSE,
  ksubset = NULL,
  include.legend = TRUE,
  do.grid = TRUE,
  ...
)
```

Arguments

bmao	a 'bma' object (cf. bms)
exact	if TRUE, then the posterior model distribution is based on the best models of bmao and their marginal likelihoods; if FALSE (default) then the distribution is based on all encountered models and their MCMC frequencies (cf. 'Details' in coef.bma)
ksubset	integer vector detailing for which model sizes the plot should be done
include.legend	if TRUE, a small legend is included via the low-level command legend
do.grid	if TRUE, a grid is added to the plot (with a simple grid()).
...	parameters passed on to matplot with sensible defaults

Value

As a default, plotModelsize plots the posterior model size distribution as a blue line, and the prior model distribution as a dashed red line.
In addition, it returns a list with the following elements:

mean	The posterior expected value of model size
var	The variance of the posterior model size distribution
dens	A vector detailing the posterior model size distribution from model size 0 (the first element) to K (the last element)

See Also

See also bms, image.bma, density.bma, plotConv

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
mm=bms(datafls,burn=1500, iter=5000, nmodel=200,mprior="fixed",mprior.size=6)

#plot Nb.1 based on aggregate results
postdist= plotModelsize(mm)

#plot based only on 30 best models
plotModelsize(mm[1:30],exact=TRUE,include.legend=FALSE)

#plot based on all best models, but showing distribution only for model sizes 1 to 20
plotModelsize(mm,exact=TRUE,ksubset=1:20)

# create a plot similar to plot Nb. 1
plot(postdist$dens,type="l")
lines(mm$mprior.info$mp.Kdist)
```

pmp.bma *Posterior Model Probabilities*

Description

Returns the posterior model probabilites for the best models encountered by a 'bma' object

Usage

```
pmp.bma(bmao, oldstyle = FALSE)
```

Arguments

bmao	A bma object (see argument nmodel in bms), alternatively an object of class topmod
oldstyle	For normal use, leave this at FALSE. It is an argument for compatibility with older BMS versions - see section 'Notes'

Details

A call to bms with an MCMC sampler (e.g. bms(datafls,mcmc="bd",nmodel=100) uses a Metropolis-Hastings algorithm to sample through the model space - and the frequency of how often models are drawn converges to the distribution of their posterior marginal likelihoods. While sampling, each 'bma' object stores the best models encountered by its sampling chain with their marginal likelihood and their MCMC frequencies.

pmp.bma then allows for comparing the posterior model probabilities (PMPs) for the two different methods, similar to plotConv. It calculates the PMPs based on marginal likelihoods (first column) and the PMPs based on MCMC frequencies (second column) for the best x models stored in the bma object.

The correlation of the two columns is an indicator of how well the MCMC sampler has converged to the actual PMP distribution - it is therefore also given in the output of summary.bma.

The second column is slightly different in case the bms argument mcmc was set to mcmc="enumeration": In this case, the second column is also based on marginal likelihoods. The correlation between the two columns is therefore one.

Value

the result is a matrix, its row names describe the model binaries
There are two columns in the matrix:

PMP (Exact)	posterior model probabilities based on the posterior likelihoods of the best models in bmao
PMP (MCMC)	posterior model probabilities of the best models in bmao based on their MCMC frequencies, relative to all models encountered by bmao - see 'Details'

Note

The second column thus shows the PMPs of the best models relative to all models the call to bms has sampled through (therefore typically the second column adds up to less than one). The first column relates to the likelihoods of the best models, therefore it would add up to 1. In order estimate for their marginal likelihoods with respect to the other models (the ones not retained in the best models), these PMP aadding up to one are multiplied with the sum of PMP of the best models accroding to MCMC frequencies. Therefore, the two columns have the same column sum.

CAUTION: In package versions up to BMS 0.2.5, the first column was indeed set always equal to one. This behaviour can still be mimicked by setting oldstyle=TRUE.

See Also

plotConv for plotting pmp.bma, pmpmodel to obtain the PMP for any individual model, bms for sampling bma objects

Check http://bms.zeugner.eu for additional help.

Examples

```
## sample BMA for growth dataset, MCMC sampler
data(datafls)
mm=bms(datafls[,1:10],nmodel=20, mcmc="bd")

## mmodel likelihoods and MCMC frequencies of best 20 models
print(mm$topmod)

pmp.bma(mm)
#first column: posterior model prob based on model likelihoods,
#  relative to best models in 'mm'
#second column: posterior model prob based MCMC frequencies,
#  relative to all models encountered by 'mm'

#consequently, first column adds up to one
#second column shows how much of the sampled model space is
# contained in the best models
colSums(pmp.bma(mm))


#correlation betwwen the two shows how well the sampler converged
cor(pmp.bma(mm)[,1],pmp.bma(mm)[,2])

#is the same as given in summary.bma
summary(mm)["Corr PMP"]

#plot the two model probabilites
plotConv(mm)

#equivalent to the following chart
plot(pmp.bma(mm)[,2], type="s")
lines(pmp.bma(mm)[,1],col=2)
```

```
#moreover, note how the first column is constructed
liks=exp(mm$top$lik())
liks/sum(liks)
pmp.bma(mm)[,1] #these two are equivalent



#the example above does not converge well,
#too few iterations and best models
# this is already better, but also not good
mm=bms(datafls[,1:10],burn=2000,iter=5000,nmodel=200)


# in case the sampler has been 'enumeration' instead of MCMC,
# then both matrix columns are of course equivalent
mm=bms(datafls[,1:10],nmodel=512,mcmc="enumeration")
cor(pmp.bma(mm)[,1],pmp.bma(mm)[,2])
colSums(pmp.bma(mm))
```

`pmpmodel` *Posterior Model Probability for any Model*

Description

Returns the posterior model probability for any model based on bma results

Usage

```
pmpmodel(bmao, model = numeric(0), exact = TRUE)
```

Arguments

`bmao`	A bma object as created by `bms`.
`model`	A model index - either variable names, or a logical with model binaries, or the model hexcode (cf. `hex2bin`, or a numeric with positions of the variables to be included.
`exact`	If `TRUE`, then the resulting PMP is based on analytical model likelihoods (works for any model). If `FALSE`, the the resulting PMP is derived from MCMC frequencies (works only for the null and fullmodel, as well as for models contained in `bmao`'s topmod object. If `bmao` is based on enumeration (cf. argument `mcmc` in `bms`, then `exact` does not matter.

Details

If the model as provided in `model` is the null or the full model, or is contained in `bmao`'s topmod object (cf. argument `nmodel` in `bms`),
then the result is the same as in `pmp.bma`.
If not and `exact=TRUE`, then `pmpmodel` estimates the model based on comparing its marginal likelihood (times model prior) to the likelihoods in the `topmod` object and multiplying by their sum of PMP according to MCMC frequencies,

Value

A scalar with (an estimate of) the posterior model probability for `model`

See Also

`pmp.bma` for similar functions

Check `http://bms.zeugner.eu` for additional help.

Examples

```
## sample BMA for growth dataset, enumeration sampler
data(datafls)
mm=bms(datafls[,1:10],nmodel=5)

#show the best 5 models:
pmp.bma(mm)
#first column: posterior model prob based on model likelihoods,
#second column: posterior model prob based MCMC frequencies,

### Different ways to get the same result: #########

#PMP of 2nd-best model (hex-code representation)
pmpmodel(mm,"00c")

#PMP of 2nd-best model (binary representation)
incls=as.logical(beta.draws.bma(mm)[,2])
pmpmodel(mm,incls)

#PMP of 2nd-best model (via variable names)
#names of regressors in model "00c":
names(datafls[,2:10])[incls]
pmpmodel(mm,c("SubSahara", "LatAmerica"))

#PMP of 2nd-best model (via positions)
pmpmodel(mm,c(6,7))

####PMP of another model #########
pmpmodel(mm,1:5)
```

post.var *Posterior Variance and Deviance*

Description

Returns posterior residual variance, deviance, or pseudo R-squared, according to the chosen prior structure

Usage

```
post.var(object, exact = FALSE, ...)
```

Arguments

object	A bma object (as produced by bms) or a zlm object.
exact	When exact=FALSE, then deviance will be based on MCMC frequencies, if exact=TRUE then it will be based on analytical posterior model probabilities - cf. argument exact in coef.bma.
...	further arguments passed to or from other methods

Details

post.var: Posterior residual variance as according to the prior definitions contained in object
post.pr2: A pseudo-R-squared corresponding to unity minus posterior variance over dependent variance.
deviance.bma: returns the deviance of a bma model as returned from bms.
deviance.zlm: returns the deviance of a zlm model.

See Also

bms for creating bma objects and priors, zlm object.

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)

mm=bms(datafls[,1:10])
deviance(mm)/nrow(datafls) # is equivalent to
post.var(mm)

post.pr2(mm) # is equivalent to
1 - post.var(mm) / ( var(datafls[,1])*(1-1/nrow(datafls)) )
```

`pred.density` *Predictive Densities for bma Objects*

Description

Predictive densities for conditional forecasts

Usage

```
pred.density(object, newdata = NULL, n = 300, hnbsteps = 30, ...)
```

Arguments

`object`	a bma object - see `bms`, alternativel a `zlm` object
`newdata`	A data.frame, matrix or vector containing variables with which to predict.
`n`	The integer number of equally spaced points at which the density is to be estimated.
`hnbsteps`	The number of numerical integration steps to be used in case of a hyper-g prior (cf. argument `g` in `bms`). Increase this number to increase accuracy. Must be an even integer.
`...`	arguments to be passed on to `plot.density`.

Details

The predictive density is a mixture density based on the `nmodels` best models in a `bma` object (cf. `nmodel` in `bms`).
The number of 'best models' to retain is therefore vital and should be set quite high for accuracy.

Value

`pred.density` returns a list of class `pred.density` with the following elements

`densities()`	a list whose elements each contain the estimated density for each forecasted observation
`fit`	a vector with the expected values of the predictions (the 'point forecasts')
`std.err`	a vector with the standard deviations of the predictions (the 'standard errors')

`dyf(realized.y, predict_index=NULL)`
Returns the densities of realized response variables provided in `realized.y`.
If `realized.y` is a matrix, then each row corresponds to a forecast observation in `newdata`
if not left empty, `predict.index` specifies to which observations in newdata the realized.y should apply

`lps(realized.y, predict_index=NULL)`
Computes the log predictive score for the response varaible provided in `realized.y` (cf. `lps.bma`) -
Note that the LPS equals minus the mean of the logarithmized results from `dyf`)

`plot((x, predict_index = NULL, addons = "eslz", realized.y = NULL, addons.lwd = 1.5, ...)`
: the same as `plot.pred.density`

`n`
: The number of equally spaced points for which the density (under `densities()` was computed.

`nmodel`
: The number of best models predictive densities are based upon.

`call`
: the call that created this `pred.density` object

Note

In BMS version 0.3.0, `pred.density` may only cope with built-in `gpriors`, not with any user-defined priors.

See Also

`predict.bma` for simple point forecasts, `plot.pred.density` for plotting predictive densities, `lps.bma` for calculating the log predictive score independently, `quantile.pred.density` for extracting quantiles

Check `http://bms.zeugner.eu` for additional help.

Examples

```
data(datafls)
mm=bms(datafls,user.int=FALSE)

#predictive densityfor two 'new' data points
pd=pred.density(mm,newdata=datafls[1:2,])


#fitted values based on best models, same as predict(mm, exact=TRUE)
pd$fit

#plot the density for the first forecast observation
plot(pd,1)

# the same plot ' naked'
plot(pd$densities()[[1]])


#predict density for the first forecast observation if the dep. variable is 0
pd$dyf(0,1)

#predict densities for both forecasts for the realizations 0 and 0.5
pd$dyf(rbind(c(0,.5),c(0,.5)))

# calc. Log Predictive Score if both forecasts are realized at 0:
lps.bma(pd,c(0,0))
```

predict.bma *Predict Method for bma Objects*

Description

Expected value of prediction based on 'bma' object

Usage

```
## S3 method for class 'bma'
predict(object, newdata = NULL, exact = FALSE, topmodels = NULL, ...)
```

Arguments

object	a bma object - see bms
newdata	An optional data.frame, matrix or vector containing variables with which to predict. If omitted, then (the expected values of) the fitted values are returned.
exact	If FALSE (default), then prediction is based on all models (i.e. on their MCMC frequencies in case the bms parameter mcmc was set to an mcmc sampler. If TRUE, then prediction is based on analytical likelihoods of the best models retained in object - cf. bms parameter nmodel.
topmodels	index of the models with whom to predict: for instance, topmodels=1 predicts based solely on the best model, whereas topmodels=1:5 predicts based on a combination of the five best models. Note that setting topmodels triggers exact=TRUE.
...	further arguments passed to or from other methods.

Value

A vector with (expected values of) fitted values.

See Also

coef.bma for obtaining coefficients, bms for creating bma objects, predict.lm for a comparable function

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
mm=bms(datafls,user.int=FALSE)

predict(mm) #fitted values based on MCM frequencies
predict(mm, exact=TRUE) #fitted values based on best models

predict(mm, newdata=1:41) #prediction based on MCMC frequencies
```

```
predict(mm, newdata=datafls[1,], exact=TRUE) #prediction based on a data.frame

# the following two are equivalent:
predict(mm, topmodels=1:10)
predict(mm[1:10], exact=TRUE)
```

`predict.zlm` *Predict Method for zlm Linear Model*

Description

Expected value (And standard errors) of predictions based on 'zlm' linear Bayesian model under Zellner's g prior

Usage

```
## S3 method for class 'zlm'
predict(object, newdata = NULL, se.fit = FALSE, ...)
```

Arguments

`object`	a zlm linear model object - see `zlm`
`newdata`	An optional data.frame, matrix or vector containing variables with which to predict. If omitted, then (the expected values of) the fitted values are returned.
`se.fit`	A switch indicating if the standard deviations for the predicted varaibles are required.
`...`	further arguments passed to or from other methods.

Value

A vector with (expected values of) fitted values.
If `se.fit` is `TRUE`, then the output is a list with the following elements:

`fit`	a vector with the expected values of fitted values
`std.err`	a vector with the standard deviations of fitted values
`se.fit`	a vector with the standard errors without the residual scale akin to `se.fit` in `predict.lm`
`residual.scale`	The part from the standard deviations that involves the identity matrix. Note that `sqrt(se.fit^2+residual.scale^2)` yields `std.err`.

See Also

`bms` for creating zlm objects, `predict.lm` for a comparable function, `predict.bma` for predicting with bma objects

Check `http://bms.zeugner.eu` for additional help.

Examples

```
data(datafls)
mm=zlm(datafls,g="EBL")

predict(mm) #fitted values
predict(mm, newdata=1:41) #prediction based on a 'new data point'

#prediction based on a 'new data point', with 'standard errors'
predict(mm, newdata=datafls[1,], se.fit=TRUE)
```

`print.topmod` *Printing topmod Objects*

Description

Print method for objects of class 'topmod', typically the best models stored in a 'bma' object

Usage

```
## S3 method for class 'topmod'
print(x, ...)
```

Arguments

`x`	an object of class 'topmod' - see `topmod`
`...`	additional arguments passed to `link{print}`

Details

See `pmp.bma` for an explanation of likelihood vs. MCMC frequency concepts

Value

if `x` contains more than one model, then the function returns a 2-column matrix:

`Row Names`	show the model binaries in hexcode
`Column 'Marg.Log.Lik'`	shows the marginal log-likelihoods of the models in `x`
`Column 'MCMC Freq'`	shows the MCMC frequencies of the models in `x`

if `x` contains only one model, then more detailed information is shown for this model:

`first line`	'Model Index' provides the model binary in hexcode, 'Marg.Log.Lik' its marginal log likelihood, 'Sampled Freq.' how often it was accepted (function `ncount()` in `topmod`)

`Estimates` first column: covariate indices included in the model, second column: posterior expected value of the coefficients, third column: their posterior standard deviations (excluded if no coefficients were stored in the topmod object - cf. argument `bbeta` in `topmod`)

`Included Covariates` the model binary

`Additional Statistics` any custom additional statistics saved with the model

See Also

`topmod` for creating topmod objects, `bms` for their typical use, `pmp.bma` for comparing posterior model probabilities

Check `http://bms.zeugner.eu` for additional help.

Examples

```
# do some small-scale BMA for demonstration
data(datafls)
mm=bms(datafls[,1:10],nmodel=20)

#print info on the best 20 models
print(mm$topmod)
print(mm$topmod,digits=10)

#equivalent:
cbind(mm$topmod$lik(),mm$topmod$ncount())



#now print info only for the second-best model:
print(mm$topmod[2])

#compare 'Included Covariates' to:
topmodels.bma(mm[2])

#and to
as.vector(mm$topmod[2]$bool_binary())
```

---

quantile.density *Extract Quantiles from 'density' Objects*

Description

Quantiles for objects of class "density", "pred.density" or "coef.density"

Usage

```
## S3 method for class 'density'
quantile(x, probs = seq(0.25, 0.75, 0.25), names = TRUE, normalize = TRUE, ...)

## S3 method for class 'coef.density'
quantile(x, probs = seq(0.25, 0.75, 0.25), names = TRUE, ...)

## S3 method for class 'pred.density'
quantile(x, probs = seq(0.25, 0.75, 0.25), names = TRUE, ...)
```

Arguments

x	a object of class pred.density, coef.density, density, or a list of densities.
probs	numeric vector of probabilities with values in [0,1] - elements very close to the boundaries return Inf or -Inf
names	logical; if TRUE, the result has a names attribute, resp. a rownames and colnames attributes. Set to FALSE for speedup with many probs.
normalize	logical; if TRUE then the values in x$y are multiplied with a factor such that their integral is equal to one.
...	further arguments passed to or from other methods.

Details

The methods quantile.coef.density and quantile.pred.density both apply quantile.density to densities nested with object of class coef.density or pred.density.
The function quantile.density applies generically to the built-in class density (as least for versions where there is no such method in the pre-configured packages).
Note that quantile.density relies on trapezoidal integration in order to compute the cumulative densities necessary for the calculation of quantiles.

Value

If x is of class density (or a list with exactly one element), a vector with quantiles.
If x is a list of densities with more than one element (e.g. as resulting from pred.density or coef.density), then the output is a matrix of quantiles, with each matrix row corresponding to the respective density.

Author(s)

Stefan Zeugner

See Also

quantile.default for a comparable function, pred.density and density.bma for the BMA-specific objects.

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
mm = bms(datafls[1:70,], user.int=FALSE)

#predict last two observations with preceding 70 obs:
pmm = pred.density(mm, newdata=datafls[71:72,], plot=FALSE)
#'standard error' quantiles
quantile(pmm, c(.05, .95))

#Posterior density for Coefficient of "GDP60"
cmm = density(mm, reg="GDP60", plot=FALSE)
quantile(cmm, probs=c(.05, .95))


#application to generic density:
dd1 = density(rnorm(1000))
quantile(dd1)

## Not run:
#application to list of densities:
quantile.density( list(density(rnorm(1000)), density(rnorm(1000))) )

## End(Not run)
```

`summary.zlm` *Summarizing Linear Models under Zellner's g*

Description

summary method for class "`zlm`"

Usage

```
## S3 method for class 'zlm'
summary(object, printout = TRUE, ...)
```

Arguments

`object`	an object of class `zlm`: see "Examples" below
`printout`	If `TRUE` (default, then information is printed to console in a neat form
`...`	further arguments passed to or from other methods

Details

`summary.zlm` prints out coefficients expected values and their standard deviations, as well as information on the gprior and the log marginal likelihood. However, it invisibly returns a list with elements as described below:

Value

A list with the following elements

residuals	The expected value of residuals from the model
coefficients	The posterior expected values of coefficients (including the intercept)
coef.sd	Posterior standard deviations of the coefficients (the intercept SD is NA, since an improper prior was used)
gprior	The g prior as it has been submitted to object
E.shrinkage	the shrinkage factor $g/(1+g)$, respectively its posterior expected value in case of a hyper-g prior
SD.shrinkage	(Optionally) the shrinkage factor's posterior standard deviation (in case of a hyper-g prior)
log.lik	The log marginal likelihood of the model

Author(s)

Stefan Zeugner

See Also

zlm for creating zlm objects, link{summary.lm} for a similar function on OLS models

See also http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)

#simple example
foo = zlm(datafls)
summary(foo)

sfoo = summary(foo,printout=FALSE)
print(sfoo$E.shrinkage)
```

---

topmod *Topmodel Object*

Description

Create or use an updateable list keeping the best x models it encounters (for advanced users)

Usage

```
topmod(
  nbmodels,
  nmaxregressors = NA,
  bbeta = FALSE,
  lengthfixedvec = 0,
  liks = numeric(0),
  ncounts = numeric(0),
  modelbinaries = matrix(0, 0, 0),
  betas = matrix(0, 0, 0),
  betas2 = matrix(0, 0, 0),
  fixed_vector = matrix(0, 0, 0)
)
```

Arguments

nbmodels	The maximum number of models to be retained by the topmod object
nmaxregressors	The maximum number of covariates the models in the topmod object are allowed to have
bbeta	if bbeta=TRUE, then first and second moments of model coefficients are stored in addition to basic model statistics (Note: if bbeta<0 then only the first moments are saved)
lengthfixedvec	The length of an optional fixed vector adhering to each model (for instance R-squared, etc). If lengthfixedvec=0 then no additonal fixed vector will be stored.
liks	optional vector of log-likelihoods to initialize topmod object with (length must be <=nbmodels) - see example below
ncounts	optional vector of MCMC frequencies to initialize topmod object with (same length as liks) - see example below
modelbinaries	optional matrix whose columns detail model binaries to initialize topmod object with (same nb columns as liks, nb rows as nmaxregressors) - see example below
betas	optional matrix whose columns are coefficients to initialize topmod object with (same dimensions as modelbinaries) - see example below
betas2	optional matrix whose columns are coefficients' second moments to initialize topmod object with (same dimensions as modelbinaries) - see example below
fixed_vector	optional matrix whose columns are a fixed vector initialize topmod object with (same ncol as modelbinaries) - see example below

Details

A 'topmod' object (as created by topmod) holds three basic vectors: lik (for the (log) likelihood of models or similar), bool() for a hexcode presentation of the model binaries (cf. bin2hex) and ncount() for the times the models have been drawn.
All these vectors are sorted descendantly by lik, and are of the same length. The maximum length is limited by the argument nbmodels.

If tmo is a topmod object, then a call to tmo$addmodel (e.g. tmo$addmodel(mylik=4,vec01=c(T,F,F,T)) updates the object tmo by a model represented by vec01 (here the one including the first and fourth regressor) and the marginal (log) likelihood lik (here: 4).

If this model is already part of tmo, then its respective ncount entry is incremented by one; else it is inserted into a position according to the ranking of lik.

In addition, there is the possibility to save (the first moments of) coefficients of a model (betas) and their second moments (betas2), as well as an arbitrary vector of statistics per model (fixed_vector).

is.topmod returns TRUE if the argument is of class 'topmod'

Value

a call to topmod returns a list of class "topmod" with the following elements:

addmodel(mylik,vec01,vbeta=numeric(0),vbeta2=numeric(0),fixedvec=numeric(0))
: function that adjusts the list of models in the 'topmod' object (see Details). mylik is the basic selection criterion (usually log likelihood), vec01 is the model binary (logical or numeric) indicating which regressors are included. vbeta is a vector of length equal to sum(vec01), contianing only the non-zero coefficients (only accounted for if bbeta!=FALSE). vbeta2 is a similar vector of second moments etc. (only accounted for if bbeta=TRUE); fixedvec is an arbitrary vector of length lengthfixedvec (see above)

lik()
: A numeric vector of the best models (log) likelihoods, in decreasing order

bool()
: A character vector of hexmode expressions for the model binaries (cf. bin2hex), sorted by lik()

ncount()
: A numeric vector of MCMC frequencies for the best models (i.e. how often the respective model was introduced by addmodel)

nbmodels
: Returns the argument nbmodel

nregs
: Returns the argument nmaxregressors

bool_binary()
: Returns a matrix whose columns present the models conforming to lik() in binary form

betas()
: a matrix whose columns are the coefficients conforming to bool_binary() (Note that these include zero coefficients due to non-inclusion of covariates); Note: if bbeta=FALSE this returns an empty matrix

betas2()
: similar to betas , for the second moments of coefficients Note: if bbeta<=0, this returns an empty matrix

fixed_vector()
: The columns of this matrix return the fixed_vector statistics conforming to lik() (see Details); Note: if lengthfixedvec=0 this returns an empty matrix

Note

topmod is rather intended as a building block for programming; it has no direct application for a user of the BMS package.

See Also

the object resulting from bms includes an element of class 'topmod'

Check http://bms.zeugner.eu for additional help.

Examples

```
#standard use
  tm= topmod(2,4,TRUE,0) #should keep a  maximum two models
  tm$addmodel(-2.3,c(1,1,1,1),1:4,5:8) #update with some model
  tm$addmodel(-2.2,c(0,1,1,1),1:3,5:7) #add another model
  tm$addmodel(-2.2,c(0,1,1,1),1:3,5:7) #add it again -> adjust ncount
  tm$addmodel(-2.5,c(1,0,0,1),1:2,5:6) #add another model

  #read out
  tm$lik()
  tm$ncount()
  tm$bool_binary()
  tm$betas()

  is.topmod(tm)

  #extract a topmod oobject only containing the second best model
  tm2=tm[2]



  #advanced: should return the same result as
  #initialize
  tm2= topmod(2,4,TRUE,0, liks = c(-2.2,-2.3), ncounts = c(2,1),
          modelbinaries = cbind(c(0,1,1,1),c(1,1,1,1)), betas = cbind(0:3,1:4),
          betas2 = cbind(c(0,5:7),5:8))

  #update
  tm$addmodel(-2.5,c(1,0,0,1),1:2,5:6) #add another model

  #read out
  tm$lik()
  tm$ncount()
  tm$bool_binary()
  tm$betas()
```

---

topmod-class *Class "topmod"*

Description

An updateable list keeping the best x models it encounters in any kind of model iteration

Objects from the Class

Objects can be created by calls to topmod, or indirectly by calls to bms.

A 'topmod' object (as created by topmod) holds three basic vectors: lik (for the (log) likelihood of models or similar), bool() for a hexcode presentation of the model binaries (cf. bin2hex) and ncount() for the times the models have been drawn.
All these vectors are sorted descendantly by lik, and are of the same length. The maximum length is limited by the argument nbmodels.

If tmo is a topmod object, then a call to tmo$addmodel (e.g. tmo$addmodel(mylik=4,vec01=c(T,F,F,T)) updates the object tmo by a model represented by vec01 (here the one including the first and fourth regressor) and the marginal (log) likelihood lik (here: 4).
If this model is already part of tmo, then its respective ncount entry is incremented by one; else it is inserted into a position according to the ranking of lik.
In addition, there is the possibility to save (the first moments of) coefficients of a model (betas) and their second moments (betas2), as well as an arbitrary vector of statistics per model (fixed_vector).

Author(s)

Martin Feldkircher and Stefan Zeugner

References

http://bms.zeugner.eu

See Also

topmod to create topmod objects and a more detailed description, is.topmod to test for this class

Examples

```
tm= topmod(2,4,TRUE,0) #should keep a  maximum two models
tm$addmodel(-2.3,c(1,1,1,1),1:4,5:8) #update with some model
tm$addmodel(-2.2,c(0,1,1,1),1:3,5:7) #add another model
tm$addmodel(-2.2,c(0,1,1,1),1:3,5:7) #add it again -> adjust ncount
tm$addmodel(-2.5,c(1,0,0,1),1:2,5:6) #add another model

#read out
tm$lik()
tm$ncount()
tm$bool_binary()
tm$betas()
```

---

topmodels.bma *Model Binaries and their Posterior model Probabilities*

Description

Returns a matrix whose columns show which covariates were included in the best models in a 'bma' object. The last two columns detail posterior model probabilities.

Usage

```
topmodels.bma(bmao)
```

Arguments

bmao	an object of class 'bma' - see bma-class

Details

Each bma class (the result of bms) contains 'top models', the x models with tthe best analytical likelihood that bms had encountered while sampling

See pmp.bma for an explanation of likelihood vs. MCMC frequency concepts

Value

Each column in the resulting matrix corresponds to one of the 'best' models in bmao: the first column for the best model, the second for the second-best model, etc. The model binaries have elements 1 if the regressor given by the row name was included in the respective models, and 0 otherwise. The second-last row shows the model's posterior model probability based on marginal likelihoods (i.e. its marginal likelihood over the sum of likelihoods of all best models) The last row shows the model's posterior model probability based on MCMC frequencies (i.e. how often the model was accepted vs sum of acceptance of all models) Note that the column names are hexcode representations of the model binaries (e.g. "03" for c(0,0,0,1,0,0))

See Also

topmod for creating topmod objects, bms for their typical use, pmp.bma for comparing posterior model probabilities

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)
#sample with a limited data set for demonstration
mm=bms(datafls[,1:12],nmodel=20)

#show binaries for all
topmodels.bma(mm)

#show binaries for 2nd and 3rd best model, without the model probs
topmodels.bma(mm[2:3])[1:11,]

#access model binaries directly
mm$topmod$bool_binary()
```

`variable.names.bma` *Variable names and design matrix*

Description

Simple utilities retrieving variable names and design matrix from a bma object

Usage

```
## S3 method for class 'bma'
variable.names(object, ...)
```

Arguments

`object`	A bma object (as produced by `bms`)
`...`	further arguments passed to or from other methods

Details

All functions are bma-functions for the generic methods `variable.names`, `deviance`, and `model.frame`.

See Also

`bms` for creating bma objects

Check `http://bms.zeugner.eu` for additional help.

Examples

```
data(datafls)
bma_enum=bms(datafls[1:20,1:10])

model.frame(bma_enum) # similar to
bma_enum$arguments$X.data

variable.names(bma_enum)[-1] # is equivalent to
bma_enum$reg.names
```

`variable.names.zlm` *Variable names and design matrix*

Description

Simple utilities retrieving variable names and design matrix from a bma object

Usage

```
## S3 method for class 'zlm'
variable.names(object, ...)
```

Arguments

`object`	A bma object (as produced by `bms`)
`...`	further arguments passed to or from other methods

Details

`variable.names.zlm`: method `variable.names` for a `zlm` model.
`vcov.zlm`: the posterior variance-covariance matrix of the coefficients of a `zlm` model - cf. `vcov`
`logLik.zlm`: a `zlm` model's log-likelihood `p(y|M)` according to the implementation of the respective coefficent prior

See Also

`zlm` for creating `zlm` objects

Check `http://bms.zeugner.eu` for additional help.

Examples

```
data(datafls)

zz=zlm(datafls)
variable.names(zz)
vcov(zz)
logLik(zz)
```

zlm *Bayesian Linear Model with Zellner's g*

Description

Used to fit the Bayesian normal-conjugate linear model with Zellner's g prior and mean zero coefficient priors. Provides an object similar to the lm class.

Usage

```
zlm(formula, data = NULL, subset = NULL, g = "UIP")
```

Arguments

formula	an object of class "formula" (or one that can be coerced to that class), such as a data.frame - cf. lm
data	an optional data.frame (or one that can be coerced to that class): cf. lm
subset	an optional vector specifying a subset of observations to be used in the fitting process.
g	specifies the hyperparameter on Zellner's g-prior for the regression coefficients. g="UIP" corresponds to $g = N$, the number of observations (default); g="BRIC" corresponds to the benchmark prior suggested by Fernandez, Ley and Steel (2001), i.e $g = max(N, K^2)$, where K is the total number of covariates; g="EBL" estimates a local empirical Bayes g-parameter (as in Liang et al. (2008)); g="hyper" takes the 'hyper-g' prior distribution (as in Liang et al., 2008) with the default hyper-parameter $a = 3$; This hyperparameter can be adjusted (between $2 < a <= 4$) by setting g="hyper=2.9", for instance. Alternatively, g="hyper=UIP" sets the prior expected value of the shrinkage factor equal to that of UIP (above), g="hyper=BRIC" sets it according to BRIC

Details

zlm estimates the coefficients of the following model $y = \alpha + X\beta + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$ and X is the design matrix
The priors on the intercept α and the variance σ are improper: $alpha \propto 1$, $sigma \propto \sigma^{-1}$
Zellner's g affects the prior on coefficients: $beta \sim N(0, \sigma^2 g(X'X)^{-1})$.
Note that the prior mean of coefficients is set to zero by default and cannot be adjusted. Note moreover that zlm always includes an intercept.

Value

Returns a list of class zlm that contains at least the following elements (cf. lm):

coefficients	a named vector of posterior coefficient expected values
residuals	the residuals, that is response minus fitted values
fitted.values	the fitted mean values

rank	the numeric rank of the fitted linear model
df.residual	the residual degrees of freedom
call	the matched call
terms	the terms object used
model	the model frame used
coef2moments	a named vector of coefficient posterior second moments
marg.lik	the log marginal likelihood of the model
gprior.info	a list detailing information on the g-prior, cf. output value gprior.info in bms

Author(s)

Stefan Zeugner

References

The representation follows Fernandez, C. E. Ley and M. Steel (2001): Benchmark priors for Bayesian model averaging. Journal of Econometrics 100(2), 381–427

See also http://bms.zeugner.eu for additional help.

See Also

The methods summary.zlm and predict.lm provide additional insights into zlm output.
The function as.zlm extracts a single out model of a bma object (as e.g. created throughbms).
Moreover, lm for the standard OLS object, bms for the application of zlm in Bayesian model averaging.

Check http://bms.zeugner.eu for additional help.

Examples

```
data(datafls)

#simple example
foo = zlm(datafls)
summary(foo)

#example with formula and subset
foo2 = zlm(y~GDP60+LifeExp, data=datafls, subset=2:70) #basic model, omitting three countries
summary(foo2)
```

`zlm-class` *Class "zlm"*

Description

A list holding output from the Bayesian Linar Model under Zellner's g prior akin to class 'lm'

Objects from the Class

Objects can be created via calls to `zlm`, but indirectly also via `as.zlm`.
`zlm` estimates a Bayesian Linear Model under Zellner's g prior - its output is very similar to objects of class `lm` (cf. section 'Value')

Author(s)

Martin Feldkircher and Stefan Zeugner

References

http://bms.zeugner.eu

See Also

`zlm` and `as.zlm` for creating `zlm` objects,
`density.zlm`, `predict.zlm` and `summary.zlm` for other posterior results

Index