

Package: rethinking (via r-universe)

September 27, 2026

Type Package

Title Statistical Rethinking book package

Version 2.42

Date 2024-5-11

Author Richard McElreath

Maintainer Richard McElreath <richard_mcelreath@eva.mpg.de>

Additional_repositories <https://stan-dev.r-universe.dev/>

Imports coda, MASS, mvtnorm, loo, shape

Depends R (>= 4.0.0), cmdstanr, posterior, parallel, methods, stats,
graphics

Suggests testthat, dagitty

Description Utilities for fitting and comparing models

License GPL (>= 3)

Repository <https://staffanbetner.r-universe.dev>

Date/Publication 2024-12-03 14:39:34 UTC

RemoteUrl <https://github.com/staffanbetner/rethinking>

RemoteRef HEAD

RemoteSha 953f01a28b3d4dc0f024cc79993ff2506c578950

Contents

rethinking-package	3
Achehunting	4
AMTL	4
axis_unscale	5
bangladesh	6
Boxes	7
chainmode	8
cherry_blossoms	8
chimpanzees	10

coefstab	11
coefstab_plot	12
coerce_index	13
col.alpha	14
compare	15
contour_xyz	16
Crofoot	16
cv_quap	17
dbetabinom	18
dens	19
dgam pois	20
Dinosaurs	21
Dissertations	22
dlk jcorr	23
dmvnorm2	24
dordlogit	25
drawdag	25
dstudent	27
dzagamma2	28
dzibinom	29
dzipois	29
ensemble	30
extract.samples	31
Fish	33
foxes	33
glimmer	34
HMC2	35
Hoogland	37
Howell	38
HPDI	38
Hurricanes	39
image_xyz	40
Kline	40
KosterLeckie	41
Laffer	42
link-methods	43
log_sum_exp	45
map2stan	45
map2stan-class	53
mcreplicate	53
milk	54
Moralizing_gods	55
pairs.map2stan	56
Panda_nuts	56
plot.map2stan	57
postcheck	58
precis	58
Primates301	59

progbar	63
PrussianHorses	64
quap	65
reedfrogs	67
resample	68
Rinder	69
sample.qa.posterior	70
shade	71
sim	73
sim_train_test	74
simplehist	75
trankplot	75
Trolley	77
UFClefties	78
ulam	79
WaffleDivorce	85
WAIC	86
Wines2012	88
Index	89

rethinking-package	<i>Statistical Rethinking package</i>
--------------------	---------------------------------------

Description

This package accompanies a book and course on Bayesian data analysis, featured MAP estimation through [quap](#) and Hamiltonian Monte Carlo through [ulam](#).

Details

Package: rethinking

Type: Package

License: GPL-3

Author(s)

Richard McElreath

References

McElreath (2020). Statistical Rethinking: A Bayesian Course with Examples in R and Stan (2nd edition). CRC Press.

See Also

[quap](#), [ulam](#)

Achehunting

Human forager hunting returns data

Description

Hunting returns of individual Ache men, 1981 to 2007.

Usage

```
data(Achehunting)
```

Format

1. month : Month of record
2. day : Day of record
3. year : Year of record
4. id : Identifier of individual man
5. age : Man's age at time of record
6. kg.meat : Kilograms of meat returned from hunt
7. hours : Duration in hours of hunting trip
8. datatype : 1 if duration of trip known, 3 otherwise

References

Hill and Kintigh. 2009. *Current Anthropology* 50:369-377.

AMTL

Ante-mortem Tooth Loss Data

Description

Data from four primate genera on tooth loss and its relationship to age and sex. Used for measurement error example in the textbook.

Usage

```
data(AMTL_short)
data(AMTL)
```

Format

1. tooth_class : One of Anterior, Posterior, or Premolar
2. specimen : Unique identifier for specimen
3. num_amtl : Number of teeth missing of given class
4. sockets : number of observable sockets that could be scored for missing teeth
5. age : Estimated age of specimen at death
6. stdev_age : Assigned uncertainty of age at death
7. prob_male : Estimate of sex of specimen
8. genus : Specimen genus, one of Homo, Pan, Papio, or Pongo
9. population : Region specimen originates from

References

Gilmore, C.C. 2013. A Comparison of Antemortem Tooth Loss in Human Hunter-Gatherers and Non-human Catarrhines: Implications for the Identification of Behavioral Evolution in the Fossil Record. American Journal of Physical Anthropology. DOI: 10.1002/ajpa.22275.

Examples

```
data(AMTL)
# plot proportion lost against age
plot( d$num_amtl / d$sockets , d$age )
```

axis_unscale

Draw an axis with units on original scale

Description

When plotting a standardized or rescaled variable, this function draws the axis units on the original scale.

Usage

```
axis_unscale( side = 1, at, orig, factor, ... )
```

Arguments

side	Side for axis. 1 is bottom.
at	Locations of tick marks, in original scale values
orig	The variable on original scale. Use this when variable was standardized.
factor	Factor the original variable was multiplied by to get rescaled variable. Use this when rescaling by a reference value, for example dividing by maximum value.

Details

This function draws a plot axis with display units on original scale. The typical situation for using this is when an analysis was performed on a standardized or rescaled variable. Plotting the posterior predictions with units on the transformed scale can make interpretation difficult.

When the variable was standardized (mean subtracted and divided by standard deviation) before analysis, use the `orig` argument to point to the variable on the original scale.

When the variable was rescaled (multiplied by a factor to rescale, without relocating zero) before analysis, use the `factor` argument.

Author(s)

Richard McElreath

Examples

```
sppnames <- c( "afarensis", "africanus", "habilis", "boisei",
               "rudolfensis", "ergaster", "sapiens" )
brainvolcc <- c( 438 , 452 , 612, 521, 752, 871, 1350 )
masskg <- c( 37.0 , 35.5 , 34.5 , 41.5 , 55.5 , 61.0 , 53.5 )
d <- data.frame( species=sppnames , brain=brainvolcc , mass=masskg )
d$mass_std <- (d$mass - mean(d$mass))/sd(d$mass)
d$brain_std <- d$brain / max(d$brain)

m7.2 <- quap(
  alist(
    brain_std ~ dnorm( mu , exp(log_sigma) ),
    mu <- a + b[1]*mass_std + b[2]*mass_std^2,
    a ~ dnorm( 0.5 , 1 ),
    b ~ dnorm( 0 , 10 ),
    log_sigma ~ dnorm( 0 , 1 )
  ), data=d , start=list(b=rep(0,2)) )

plot( d$brain_std ~ d$mass_std , xaxt="n" , yaxt="n" , xlab="body mass (kg)" , ylab="brain volume (cc)" , col=range(1:2) )
axis_unscale( 1 , at=quantile(d$mass) , d$mass )
axis_unscale( 2 , at=quantile(d$brain) , factor=max(d$brain) )

mass_seq <- seq(from=-1,to=1.5,length.out=30)
mu <- link(m7.2,data=list(mass_std=mass_seq))
mu <- apply(mu,2,mean)
lines( mass_seq , mu )
```

bangladesh

Bangladesh contraceptive use data

Description

Contraceptive use data from 1934 Bangladeshi women.

Usage

```
data(bangladesh)
```

Format

1. woman : ID number for each woman in sample
2. district : Number for each district
3. use.contraception : 0/1 indicator of contraceptive use
4. living.children : Number of living children
5. age.centered : Centered age
6. urban : 0/1 indicator of urban context

References

Bangladesh Fertility Survey, 1989

Boxes

Social learning experimental data

Description

Data from (and Stan models for) a cross-cultural experiment investigating the development of social learning in children

Usage

```
data(Boxes)
data(Boxes_model)
data(Boxes_model_age)
data(Boxes_model_gender)
```

Format

1. y : Outcome, one of 1=unchosen option, 2=majority option, or 3=minority option
2. gender : Index of child's gender. 1=girl. 2=boy.
3. age : Age of each child in years
4. majority_first : whether the majority option was demonstrated first
5. culture : ID of the site, from 1 to 8

Author(s)

Richard McElreath

References

van Leeuwen et al 2018. The development of human social learning across seven societies. DOI: 10.1038/s41467-018-04468-2.

chainmode

Find mode of a continuous density estimate

Description

Returns estimated mode of a density computed from samples.

Usage

```
chainmode( chain , ... )
```

Arguments

chain	Values, e.g. sampled from a posterior via MCMC
...	Optional arguments passed to density calculation

Details

This function just finds the x value that maximizes the y density in the density estimate.

Author(s)

Richard McElreath

cherry_blossoms

Japan Cherry Blossom Historical Data

Description

Historical Series of Phenological data for Cherry Tree Flowering at Kyoto City.

Usage

```
data(cherry_blossoms)
```

Format

1. year: Year CE
2. doy: Day of year of first bloom. Day 89 is April 1. Day 119 is May 1.
3. temp: March temperature estimate
4. temp_upper: Upper 95% bound for estimate
5. temp_lower: Lower 95% bound for estimate

References

Aono and Saito 2010. International Journal of Biometeorology, 54, 211-219. Aono and Kazui 2008. International Journal of Climatology, 28, 905-914. Aono 2012. Chikyu Kankyo (Global Environment), 17, 21-29. <http://atmenv.envi.osakafu-u.ac.jp/aono/kyophenotemp4/>

Examples

```
# This code reproduces the plot on the 2nd edition cover

library(rethinking)
data(cherry_blossoms)
d <- cherry_blossoms

# spline on temp
d2 <- d[ complete.cases(d$temp) , ] # complete cases on temp

num_knots <- 30
( knot_list <- quantile( d2$year , probs=seq(0,1,length.out=num_knots) ) )

library(splines)
B <- bs(d2$year,
        knots=knot_list[-c(1,num_knots)] ,
        degree=3 , intercept=TRUE )

m1 <- quap(
  alist(
    T ~ dnorm( mu , sigma ) ,
    mu <- a + B
    a ~ dnorm(6,1),
    w ~ dnorm(0,10),
    sigma ~ dexp(1)
  ),
  data=list( T=d2$temp , B=B ) ,
  start=list( w=rep( 0 , ncol(B) ) ) )

# now spline on blossom doy
d3 <- d[ complete.cases(d$doy) , ] # complete cases on doy

knot_list <- seq( from=min(d3$year) , to=max(d3$year) , length.out=num_knots )
B3 <- t(bs(d3$year, knots=knot_list , degree=3, intercept = FALSE))

m2 <- quap(
  alist(
    Y ~ dnorm( mu , sigma ) ,
    mu <- a0 + as.vector( a
    a0 ~ dnorm(100,10),
    a ~ dnorm(0,10),
    sigma ~ dexp(1)
  ),
  data=list( Y=d3$doy , B=B3 ) , start=list(a=rep(0,nrow(B3))) )

# PLOT
```

```

blank2(w=2,h=2)

par( mfrow=c(2,1) , mgp = c(1.25, 0.25, 0), mar = c(0.75, 2.5, 0.75, 0.75) + 0.1,
      tck = -0.02, cex.axis = 0.8 )

xcex <- 1.2
xpch <- 16
xcol1 <- col.alpha(rangi2,0.3)
col_spline <- col.alpha("black",0.4)
xlims <- c(850,2000)

plot( d2$year , d2$temp , ylab="March temperature" , col=xcol1 , pch=xpch , cex=xcex , xlab="" , xlim=xlims , bty="n" )
l <- link( m1 )
li <- apply(1,2,PI,0.97)

atx <- c(900,1400,2000)
axis( 1 , at=atx , labels=paste(atx,"CE") )
axis( 2 , at=c(5,8) , labels=c("5°C","8°C") )

y <- d3$doy
y <- y - min(y)
y <- y/max(y)
blossom_col <- sapply( d3$doy , function(y) hsv(1, rbeta2(1, inv_logit(logit(0.1)+0.02*y) ,10) ,1,0.8) )
plot( NULL , cex=xcex , ylab="Day of first blossom" , xlim=xlims , bty="n" , axes=FALSE , xlab="" , ylim=range(d3$doy) )
l <- link( m2 )
li <- apply(1,2,PI,0.9)
points( d3$year , d3$doy , col=blossom_col , pch=8 , cex=xcex , lwd=2 )
shade( li , d3$year , col=grau(0.3) )

axis( 2 , at=c(90,120) , labels=c("April 1","May 1") )

```

chimpanzees

Chimpanzee prosocialty experiment data

Description

Data from behavior trials in a captive group of chimpanzees, housed in Lousiana. From Silk et al. 2005. Nature 437:1357-1359.

Usage

```
data(chimpanzees)
```

Format

1. actor : name of actor
2. recipient : name of recipient (NA for partner absent condition)

3. condition : partner absent (0), partner present (1)
4. block : block of trials (each actor x each recipient 1 time)
5. trial : trial number (by chimp = ordinal sequence of trials for each chimp, ranges from 1-72; partner present trials were interspersed with partner absent trials)
6. prosocial_left : 1 if prosocial (1/1) option was on left
7. chose_prosoc : choice chimp made (0 = 1/0 option, 1 = 1/1 option)
8. pulled_left : which side did chimp pull (1 = left, 0 = right)

Author(s)

Richard McElreath

References

Silk et al. 2005. Nature 437:1357-1359.

coefTab

Coefficient tables

Description

Returns a table of model coefficients in rows and models in columns.

Usage

```
coefTab( ... , se=FALSE , se.inside=FALSE , nobs=TRUE , digits=2 , width=7 , rotate=FALSE )
```

Arguments

...	A series of fit models, separated by commas
se	Include standard errors in table?
se.inside	Print standard errors in same cell as estimates
nobs	Print number of observations for each model?
digits	Number of digits to round numbers to
rotate	If TRUE, rows are models and columns are coefficients

Details

This function provides a way to compare estimates across models.

Author(s)

Richard McElreath

coefstab_plot

*Plots of coefficient tables***Description**

Plots coefficient tables produced by `coefstab`, clustered either by models or by parameter names.

Usage

```
coefstab_plot( x , y , pars , col.ci="black" , by.model=FALSE ,
  prob=0.95 , ... )
```

Arguments

<code>x</code>	Object produced by <code>coefstab</code>
<code>y</code>	NULL and unused. Required for compatibility with base plot
<code>pars</code>	Optional vector of parameter names or indexes to display. If missing, all parameters shown.
<code>col.ci</code>	Color to draw confidence intervals
<code>by.model</code>	Cluster estimates by model instead of by parameter (default)
<code>prob</code>	Probability mass for confidence intervals. Default is 0.95.

Details

This function plots the tabular output of `coefstab`, using a `dotchart`. By default, estimates are grouped by parameter, with a row for each model. Model's without a parameter still appear as a row, but with no estimate. By setting `by.model=TRUE`, the dotchart will instead be grouped by model, with each row being a parameter.

MAP estimates are displayed with percentile confidence (credible) intervals. Default is 95% intervals. Use `prob` to change the interval mass.

Author(s)

Richard McElreath

See Also

`coefstab`, `dotchart`

coerce_index	<i>Build and check integer index variables</i>
--------------	--

Description

These functions assist with building (`coerce_index`) and checking (`check_index`) integer index variables of the kind needed to define varying effect models.

Usage

```
coerce_index( ... )
check_index( x )
```

Arguments

<code>...</code>	A comma-separated list of variables. See details.
<code>x</code>	A vector of integers to check for contiguity

Details

Varying effect models often require index variables that begin at 1 and comprise only integers. These variables are used to lookup specific parameter values inside of the model. For example, it is common to define varying intercepts with a linear model like $a_0 + a_{id[id[i]]}$. Here the variable `id` is an index variable. It has one value for each case, defining which individual applies to that case.

When raw data exist as factors, these index variables much be converted to integers. This is trickier than it sounds, because R uses an internal integer representation for factors, levels, that can conflict with ordinary integer representations.

The function `coerce_index` deals with that complication. When the input is a single vector of factors, it returns an integer vector beginning at 1 and with contiguous values.

When the input is instead a comma-separated list of factors, it returns a list in which each factor has been converted to integers, but all levels in all factors were merged so that the same labels across factors always have the same integer values in the result. For example, suppose cases refer to dyads and there are two factors, `id1` and `id2`, that indicate which pair of individuals are present in each dyad. The labels in these variables should refer to the same individuals. Passing both simultaneously to `coerce_index` ensures that the results respect that fact.

The function `check_index` merely checks an integer vector to see if it is contiguous.

Value

For `coerce_index`, the result is either a single vector of integers (if the input was a single vector) or rather a list of vectors of integers (if the input was a list of vectors).

Author(s)

Richard McElreath

col.alpha

*Color utility functions***Description**

Functions for calculating transparent and desaturated colors.

Usage

```
col.alpha( acol , alpha = 0.2 )
col.desat( acol , amt = 0.5 )
col.dist( x , mu = 0 , sd = 1 , col="slateblue" )
grau( alpha = 0.5 )
```

Arguments

acol	A color name or RGB color
alpha	alpha transparency, where 1 means fully opaque and 0 fully transparent
amt	amount of desaturation of color to apply, where 1 means totally desaturated (grayscale)
x	A vector of values to use for calculating distances. See details below.
mu	Value (or vector of values) to use for calculating distances
sd	Standard deviation of distance function used to calculate transparency
col	A color to apply transparency to, based on distance
alpha	Transparency of black (used by grau function)

Details

These functions allow for calculating transparency and desaturation for colors. `col.alpha` makes a base color transparent, while `col.desat` makes a color have less saturation.

`col.dist` is used to make a list of transparent colors of differing alpha level. The levels are chosen based upon Gaussian distance from a chosen value, `mu`, with a chosen width of the function that determines how quickly colors become fully transparent, `sd`. For example, if `x` contains a column of data, then `col.dist` will return a vector of same length with transparency increasing as each value in `x` is distant from `mu`. This is useful for plotting data that emphasize points near some value or values.

`grau` simply returns a transparent version of black, producing effective gray values.

Author(s)

Richard McElreath

See Also

[col2rgb](#), [rgb2hsv](#), [rgb](#)

compare

Compare fit models using WAIC or DIC

Description

Returns a table of model comparison statistics, by default focused on WAIC.

Usage

```
compare( ... , n=1e3 , sort="WAIC" , func=WAIC , WAIC=TRUE , refresh=0 , log_lik="log_lik" )
ICweights( dev )
```

Arguments

...	A series of fit models, separated by commas
n	Number of samples from posterior to use in computing WAIC/DIC
sort	Sort table by ascending values in named column
func	Function to use in computing criteria for comparison
WAIC	Deprecated: If TRUE, uses func for comparison. If FALSE, uses DIC.
refresh	Progress display update interval. 0 suppresses display.
dev	Vector of values to use in computing model weights
log_lik	Name of log likelihood list in posterior

Details

This function computes WAIC and optionally DIC values for fit models and returns a table sorted by ascending values. Each row in this table is a model, and the various columns provide WAIC, effective numbers of parameters, model weights, and standard errors.

A plot method is supported, for graphic display of the information criteria.

Value

An object of class `compareIC` with slots `output` (table of results) and `dSE` (matrix of standard errors of differences in IC between pairs of models).

Author(s)

Richard McElreath

contour_xyz	<i>Contour plot from equal length x,y,z vectors</i>
-------------	---

Description

Provides an interface to use contour by providing three equal length vectors for x, y and z coordinates.

Usage

```
contour_xyz( x , y , z , ... )
```

Arguments

- x vector of x values
- y vector of y values
- z vector of z values
- ... other parameters to pass to contour

Details

This function merely constructs a matrix suitable for contour, using x, y and z coordinates.

Author(s)

Richard McElreath

See Also

[contour](#)

Crofoot	<i>Capuchin monkey contests</i>
---------	---------------------------------

Description

Data on features and outcomes of intergroup contests among territorial groups of capuchin monkey (*Cebus capucinus*). Each row contains a single contest between two groups.

Usage

```
data(Crofoot)
```

Format

1. focal : ID of focal group
2. other : ID of other group
3. dyad : ID of specific dyad pair of groups
4. win : 1 if focal won contest, 0 if other won
5. dist_focal : Distance in meters of focal group from the center of its home range
6. dist_other : Distance in meters of other group from the center of its home range
7. n_focal : Number of individuals in focal group
8. n_other : Number of individuals in other group
9. m_focal : Number of males in focal group
10. m_other : Number of males in other group
11. f_focal : Number of females in focal group
12. f_other : Number of females in other group

References

M.C. Crofoot, I.C. Gilby, M.C. Wikelski, and R.W. Kays. 2008. PNAS 105:577–581.

cv_quap

Diagnostic trace and rank histogram plots for MCMC output

Description

Cross validation for quap model fits.

Usage

```
cv_quap( quap_model, lno = 1, pw = FALSE, cores = 1, ... )
```

Arguments

quap_model	quap model fit
lno	Number of observations to leave out in each fold
pw	Pointwise results (TRUE) or summed across folds (FALSE)
cores	Number of cores to use. If great than 1, uses mclapply with folds
...	Additional arguments to pass to quap

Details

This function constructs cross validation folds from an existing quap model fit and associated data. It then fits the model to each fold and returns either the fit to each fold (when pw=TRUE) or the summed performance across all folds.

The default is leave-one-out cross-validation, when lno=1.

Author(s)

Richard McElreath

dbetabinom

*Beta-binomial probability density***Description**

Functions for computing density and producing random samples from a beta-binomial probability distribution.

Usage

```
dbetabinom( x , size , prob , theta , shape1 , shape2 , log=FALSE )
rbetabinom( n , size , prob , theta , shape1 , shape2 )
```

Arguments

x	Integer values to compute probabilities of
size	Number of trials
prob	Average probability of beta distribution
theta	Dispersion of beta distribution
shape1	First shape parameter of beta distribution (alpha)
shape2	Second shape parameter of beta distribution (beta)
log	If TRUE, returns log-probability instead of probability
n	Number of random observations to sample

Details

These functions provide density and random number calculations for beta-binomial observations. The dbetabinom code is based on Ben Bolker's original code in the emdbook package.

Either prob and theta OR shape1 and shape2 must be provided. The two parameterizations are related by $\text{shape1} = \text{prob} * \text{theta}$, $\text{shape2} = (1 - \text{prob}) * \text{theta}$.

The rbetabinom function generates random beta-binomial observations by using both [rbeta](#) and [rbinom](#). It draws n values from a beta distribution. Then for each, it generates a random binomial observation.

Author(s)

Richard McElreath

Examples

```
## Not run:
data(reedfrogs)
reedfrogs$pred_yes <- ifelse( reedfrogs$pred=="pred" , 1 , 0 )

# map model fit
# note exp(log_theta) to constrain theta to positive reals
m <- map(
  alist(
    surv ~ dbetabinom( density , p , exp(log_theta) ),
    logit(p) <- a + b*pred_yes,
    a ~ dnorm(0,10),
    b ~ dnorm(0,1),
    log_theta ~ dnorm(1,10)
  ),
  data=reedfrogs )

# map2stan model fit
# constraint on theta is passed via constraints list
m.stan <- map2stan(
  alist(
    surv ~ dbetabinom( density , p , theta ),
    logit(p) <- a + b*pred_yes,
    a ~ dnorm(0,10),
    b ~ dnorm(0,1),
    theta ~ dcauchy(0,1)
  ),
  data=reedfrogs,
  constraints=list(theta="lower=0") )

## End(Not run)
```

dens

Density plots

Description

Convenient interface for plotting density estimates.

Usage

```
dens( x , adj=0.5 , norm.comp=FALSE , main="" , show.HPDI=FALSE , show.zero=FALSE , rm.na=TRUE , add=FAL
```

Arguments

x	Vector of values to construct density from, or data frame. If x is a data frame, then dens plots a grid of densities, one for each column in x.
adj	width of density kernel.
norm.comp	If TRUE, overlays normal density comparison.

<code>show.HPDI</code>	If a numeric value, displays HPDI of same width. For example, <code>show.HPDI=0.95</code> shows a 95 percent HPDI inside the density.
<code>show.zero</code>	If TRUE, draws a vertical line at location of zero on horizontal axis.
<code>rm.na</code>	If TRUE, removes NAs before computing density
<code>add</code>	If TRUE, overlays density on an existing plot
<code>...</code>	Other parameters to pass to <code>density</code> , which constructs the density estimates.

Details

This function merely provides a convenient interface for plotting density estimates produced by `density`. It handles both single vectors and multiple vectors, contained within a data frame.

Highest Posterior Density Intervals (HPDI) are calculated by `HPDinterval` in the `coda` package.

Author(s)

Richard McElreath

See Also

[density](#), [HPDinterval](#)

dgam pois

Gamma-Poisson probability density

Description

Functions for computing density and producing random samples from a gamma-Poisson (negative-binomial) probability distribution.

Usage

```
dgam pois( x , mu , scale , log=FALSE )
rgam pois( n , mu , scale )
```

Arguments

<code>x</code>	Integer values to compute probabilities of
<code>mu</code>	Mean of gamma distribution
<code>scale</code>	Scale parameter of gamma distribution
<code>log</code>	If TRUE, returns log-probability instead of probability
<code>n</code>	Number of random observations to sample

Details

These functions provide density and random number calculations for gamma-Poisson observations. These functions use `dnbinom` and `rnbinom` internally, but convert the parameters from the `mu` and `scale` form. The size parameter of the negative-binomial is defined by `mu/scale`, and the prob parameter of the negative-binomial is the same as `1/(1+scale)`.

Author(s)

Richard McElreath

Examples

```
## Not run:
data(Hurricanes)

# map model fit
# note exp(log_scale) to constrain scale to positive reals
m <- map(
  alist(
    deaths ~ dgamma(mu , exp(log_scale) ),
    log(mu) <- a + b*femininity,
    a ~ dnorm(0,100),
    b ~ dnorm(0,1),
    log_scale ~ dnorm(1,10)
  ),
  data=Hurricanes )

# map2stan model fit
# constraint on scale is passed via constraints list
m.stan <- map2stan(
  alist(
    deaths ~ dgamma(mu , scale ),
    log(mu) <- a + b*femininity,
    a ~ dnorm(0,100),
    b ~ dnorm(0,1),
    scale ~ dcauchy(0,2)
  ),
  data=Hurricanes,
  constraints=list(scale="lower=0"),
  start=list(scale=2) )

## End(Not run)
```

Dinosaurs

Dinosaur growth data

Description

Mass by age data for 6 species of dinosaur.

Usage

```
data(Dinosaurs)
```

Format

1. age : Estimated age of specimen at death, in years
2. mass : Estimated body mass at death, in kilograms
3. species : Species name
4. sp_id : Identification number of species

References

Erickson, Rogers, Yerby. 2001. Dinosaurian growth patterns and rapid avian growth rates. *Nature* 412:429-433.

Dissertations

Dissertation page length data

Description

Page lengths of 3037 dissertations filed between 2006 and 2014 at the University of Minnesota.

Usage

```
data(Dissertations)
```

Format

1. pages: number of pages
2. major: department/program of dissertation
3. year: year of filing

References

Data originally parsed by...

dlkjcorr

*LKJ correlation matrix probability density***Description**

Functions for computing density and producing random samples from the LKJ onion method correlation matrix distribution.

Usage

```
dlkjcorr( x , eta=1 , log=TRUE )
rlkjcorr( n , K , eta=1 )
```

Arguments

x	Matrix to compute probability density for
eta	Parameter controlling shape of distribution
K	Dimension of correlation matrix
log	If TRUE, returns log-probability instead of probability
n	Number of random matrices to sample

Details

The LKJ correlation matrix distribution is based upon work by Lewandowski, Kurowicka, and Joe. When the parameter eta is equal to 1, it defines a flat distribution of correlation matrices. When $\eta > 1$, the distribution is instead concentrated towards to identity matrix. When $\eta < 1$, the distribution is more concentrated towards extreme correlations at -1 or +1.

It can be easier to understand this distribution if we recognize that the individual correlations within the matrix follow a beta distribution defined on -1 to +1. Thus eta resembles theta in the beta parameterization with a mean μ and scale (sample size) θ .

The function `rlkjcorr` returns an 3-dimensional array in which the first dimension indexes matrices. In the event that $n=1$, it returns instead a single matrix.

Author(s)

Richard McElreath

References

Lewandowski, Kurowicka, and Joe. 2009. Generating random correlation matrices based on vines and extended onion method. *Journal of Multivariate Analysis*. 100:1989-2001.

Stan Modeling Language User's Guide and Reference Manual, v2.6.2

Examples

```

R <- rlkjcorr(n=1,K=2,eta=4)
dlkjcorr(R,4)

# plot density of correlation
R <- rlkjcorr(1e4,K=2,eta=4)
dens( R[,1,2] )

# visualize 3x3 matrix
R <- rlkjcorr(1e3,K=3,eta=2)
plot( R[,1,2] , R[,1,3] , col=col.alpha("black",0.2) , pch=16 )

```

dmvnorm2

*Multivariate Gaussian probability density***Description**

This is an alternative parameterization of the ordinary multivariate Gaussian probability density.

Usage

```

dmvnorm2( x , Mu , sigma , Rho , log=FALSE )
rmvnorm2( n , Mu=rep(0,length(sigma)) , sigma=rep(1,length(Mu)) ,
          Rho=diag(length(Mu)) , method="chol" )

```

Arguments

x	Values to compute densities of
Mu	Mean vector
sigma	Vector of standard deviations
Rho	Correlation matrix
log	If TRUE, returns log-density instead of density
n	Number of random observations to sample

Details

These functions merely compose the variance-covariance matrix from separate standard deviation and correlation matrix arguments. They then use `dmvnorm` and `rmvnorm` from the `mvtnorm` package to perform calculations.

Author(s)

Richard McElreath

Examples

```

dmvnorm2( c(1,0) , Mu=c(0,0) , sigma=c(1,1) , Rho=diag(2) )
rmvnorm2( 10 , Mu=c(1,2) )

```

dordlogit	<i>Ordered categorical log-odds probability density</i>
-----------	---

Description

Functions for computing density, cumulative probability and producing random samples from an ordered categorical probability density.

Usage

```
dordlogit( x , phi , a , log=FALSE )
pordlogit( x , phi , a , log=FALSE )
rordlogit( n , phi , a )
```

Arguments

x	Integer values to compute densities or probabilities of
a	Vector of log-odds intercepts
phi	Linear model of log-odds
log	If TRUE, returns log-probability instead of probability

Details

These functions provide for common density calculations for the ordered categorical probability density commonly known as an "ordered logit" or "ordered logistic." This is the same probability density assumed by the `polr` function (library MASS).

Author(s)

Richard McElreath

drawdag	<i>Plot Directed Acyclic Graphs (dagitty)</i>
---------	---

Description

A fancier version of dagitty's plot function, as well as a way to show open paths given a conditioning set.

Usage

```
drawdag( x , col_arrow="black" , col_segment="black" , col_labels="black" , cex=1 ,
  lwd=1.5 , goodarrow=TRUE , xlim , ylim , shapes , col_shapes , radius=3 ,
  add=FALSE , ... )

drawopenpaths( x , Z=list() , col_arrow="red" , ... )
```

Arguments

<code>x</code>	A dagitty graph
<code>col_arrow</code>	Color or vector of colors for the graph arrows
<code>col_segment</code>	Color or vector of colors for the graph segments
<code>col_labels</code>	Color or vector of colors for the graph text labels
<code>cex</code>	Size of text labels
<code>lwd</code>	Width of arrow lines
<code>goodarrow</code>	Use Arrow from shape package to draw arrows
<code>xlim</code>	Optional plot limits
<code>ylim</code>	Optional plot limits
<code>shapes</code>	A named list of variables with one of "c" for an open circle or "fc" for a filled circle
<code>col_shapes</code>	A named list of colors to correspond to the shapes list
<code>radius</code>	Radius of shapes circles
<code>add</code>	If TRUE, draw over existing DAG in active plot
<code>Z</code>	List of variables to condition on when computing open paths
<code>...</code>	Optional arguments to pass to other functions

Details

drawdag is a modified version of `plot.dagitty` but with additional stylistic options. By default, it draws arrows in black and with thicker line width. It also uses the nicer arrows drawn by `Arrows` in the `shape` package. If the DAG doesn't already have `coordinates`, then `graphLayout` is called to provide them.

drawopenpaths uses `dagitty::paths` to compute and then overdraw open paths, given an exposure and outcome variable. It uses drawdag to perform the drawing. Requires that the DAG already be displayed. Only open paths are drawn in the overlay color.

Author(s)

Richard McElreath

Examples

```
## Not run:

ex1 <- dagitty("dag {
  X [exposure]
  Y [outcome]
  U [unobserved]
  Z -> X -> Y
  X <- U -> Y
}")
coordinates(ex1) <- list( x=c(Z=0,X=1,Y=1,U=0) , y=c(Z=0,U=0.5,X=0,Y=1) )
drawdag( ex1 )
```

```
# example of drawing open paths
drawdag( ex1 )
drawopenpaths( ex1 ) # open backdoor

drawdag( ex1 , col_arrow="gray" )
drawopenpaths( ex1 , Z=list("U") ) # closed backdoor

## End(Not run)
```

dstudent

*Non-standard Student's t distribution***Description**

Functions for computing density and producing random samples from a non-standardized Student's t distribution.

Usage

```
dstudent( x, nu = 2, mu = 0, sigma = 1, log = FALSE )
rstudent( n, nu = 2, mu = 0, sigma = 1 )
```

Arguments

x	Values to compute densities of
nu	Degrees of freedom (tail shape)
mu	Location of distribution (mean)
sigma	Scale of distribution
log	If TRUE, returns log-density instead of density
n	Number of random observations to sample

Details

These functions provide density and random number calculations for Student's t distribution, translated and scaled by mean μ and scale σ . Note that σ is not the distribution's standard deviation, unless ν is very large.

Author(s)

Richard McElreath

Examples

```
## Not run:
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce
d$A <- scale( d$MedianAgeMarriage )
d$D <- scale( d$Divorce )
d$M <- scale( d$Marriage )

m5.3b <- quap(
  alist(
    D ~ dstudent( 2 , mu , sigma ) ,
    mu <- a + bM*M + bA*A ,
    a ~ dnorm( 0 , 0.2 ) ,
    bM ~ dnorm( 0 , 0.5 ) ,
    bA ~ dnorm( 0 , 0.5 ) ,
    sigma ~ dexp( 1 )
  ) , data = d )

## End(Not run)
```

dzagamma2

Zero-augmented gamma probability density

Description

Function for computing density from a zero-augmented gamma probability distribution.

Usage

```
dzagamma2( x , prob , mu , scale , log=FALSE )
```

Arguments

x	Values to compute densities for
prob	Probability of a zero
mu	Mean of gamma distribution
scale	Scale parameter (same as scale in dgamma)
log	If TRUE, returns log-density instead of density

Details

This distribution is defined as a finite mixture of zeros and strictly positive gamma distributed values, where prob determines the weight of the zeros. As such, the probability of a zero is prob, and the probability of a non-zero value x is $(1-\text{prob}) \cdot \text{dgamma}(x, \text{mu}/\text{scale}, \text{scale})$.

Author(s)

Richard McElreath

dzibinom	<i>Zero-inflated binomial probability density</i>
----------	---

Description

Functions for computing density and producing random samples from a zero-inflated binomial probability distribution.

Usage

```
dzibinom( x , p_zero , size , prob , log=FALSE )
rzibinom( n , p_zero , size , prob )
```

Arguments

x	Integer values to compute densities or probabilities of
p_zero	Probability of a zero from bernoulli process
size	Number of binomial trials
prob	Probability of success in binomial trial
log	If TRUE, returns log-probability instead of probability

Details

These functions provide density and random number calculations for zero-inflated binomial observations. This distribution is defined as a finite mixture of zeros and binomial values, where `p_zero` determines the weight of the pure zeros. As such, the probability of a zero is $p_zero + (1-p_zero)(1-prob)^{size}$. And the probability of a non-zero value x is $(1-p_zero) \text{choose}(size, x) prob^x (1-prob)^{(size-x)}$.

`dzibinom` does its calculations in a way meant to reduce numerical issues with probabilities close to 0 and 1. As a result, it's not very fast.

Author(s)

Richard McElreath

dzipois	<i>Zero-inflated Poisson probability density</i>
---------	--

Description

Functions for computing density and producing random samples from a zero-inflated Poisson probability distribution.

Usage

```
dzipois( x , p , lambda , log=FALSE )
rzipois( n , p , lambda )
```

Arguments

x	Integer values to compute densities or probabilities of
p	Probability of a zero from bernoulli process
lambda	Poisson rate parameter (mean)
log	If TRUE, returns log-probability instead of probability

Details

These functions provide density and random number calculations for zero-inflated Poisson observations. This distribution is defined as a finite mixture of zeros and Poisson values, where p determines the weight of the pure zeros. As such, the probability of a zero is $p + (1-p)\exp(-\lambda)$. And the probability of a non-zero value x is $(1-p)\lambda^x \exp(-\lambda)/x!$.

`dzipois` does its calculations in a way meant to reduce numerical issues with probabilities close to 0 and 1. As a result, it's not very fast.

Author(s)

Richard McElreath

ensemble	<i>Simulate an ensemble of posterior predictions</i>
----------	--

Description

Uses `link` and `sim` for a list of `map` or `map2stan` model fits to construct Akaike weighted ensemble of predictions.

Usage

```
ensemble( ... , data , n=1e3 , func=WAIC , weights , WAIC=TRUE , refresh=0 ,
          replace=list() , do_link=TRUE , do_sim=TRUE )
```

Arguments

...	<code>map</code> or <code>map2stan</code> models
data	Optional data to compute predictions over, as in <code>link</code> and <code>sim</code>
n	Number of samples to draw from posterior for each model
func	Function to use in computing criterion for model weights
weights	Optional vector of weights to use. If present, <code>func</code> is ignored.

WAIC	Deprecated: If TRUE, use func to compute weights. Otherwise tries to use DIC.
refresh	Progress update refresh interval. 0 suppresses output.
replace	Optional named list with replacement posterior samples. Used for maginalizing over random effects, for example. See example in link .
do_link	If TRUE, compute and return link results
do_sim	If TRUE, compute and return sim results

Details

This function calls [link](#) and [sim](#) for each fit model given as input. The results are then combined into ensemble link and simulation output, where samples from each model are represented in proportion to the Akaike weights. Akaike weights are calculated by [compare](#), using func, unless an explicit vector weights is provided. The values in weights will be normalized to sum to one, if they do not already.

Note that no averaging is done by this function. Parameters are not averaged, and predictions are not averaged.

Author(s)

Richard McElreath

See Also

[link](#), [sim](#), [compare](#)

extract.samples	<i>Collect posterior or prior samples from a map or map2stan model</i>
-----------------	--

Description

Extracts or draw samples from fit models.

Usage

```
extract.samples( object , n=10000 , pars , ... )
extract.prior( object , n=1000 , pars , ... )
```

Arguments

object	Fit model to extract samples from
n	Number of samples to simulate
pars	Character vector of parameters to return
...	Other parameters to pass to descendent functions (when defined)

Details

Use `extract.samples` and `extract.prior` to return lists of samples for posterior and prior distributions, respectively. Methods for `extract.samples` are provided for `map`, `map2stan`, and `stanfit` objects. Methods for `extract.prior` are provided for `map` and `map2stan` objects.

For `map2stan` and `stanfit` models, `extract.samples` returns cleaned samples already contained in the object. These samples are cleaned of dimension attributes and the `lp__`, `dev`, and `log_lik` traces that are used internally. For `map` and other types, it uses the variance-covariance matrix and coefficients to define a multivariate Gaussian posterior to draw `n` samples from.

`extract.prior` must simulate draws using the model definition. It attempts to return samples in the same list structure as posterior samples. This makes prior-posterior comparison easier. See examples below.

Value

A named list (for `map2stan` and `stanfit`) or `data.frame` containing samples for each parameter in the posterior/prior distribution.

Author(s)

Richard McElreath

See Also

[mvrnorm](#)

Examples

```
data(chimpanzees)

d <- list(
  pulled_left = chimpanzees$pulled_left ,
  prosoc_left = chimpanzees$prosoc_left ,
  condition = chimpanzees$condition ,
  actor = as.integer( chimpanzees$actor ) ,
  blockid = as.integer( chimpanzees$block )
)

m <- map(
  alist(
    pulled_left ~ dbinom(1,theta),
    logit(theta) <- a + aj[actor] + bp*prosoc_left + bpc*condition*prosoc_left,
    aj[actor] ~ dnorm( 0 , 1 ),
    a ~ dnorm(0,2),
    bp ~ dnorm(0,1),
    bpc ~ dnorm(0,1)
  ) ,
  data=d )

prior <- extract.prior(m,n=1e4)
post <- extract.samples(m)
```

```
ps <- par("bty")
par(bty="n")
plot( precis(prior,2) , col.ci="gray" , xlim=c(-3,3.5) , bty="n" )
plot( precis(post,2) , add=TRUE , pch=16 )
par(bty=ps)
```

Fish

Fishing data

Description

Fishing data from visitors to a national park.

Usage

```
data(Fish)
```

Format

1. fish_caught : Number of fish caught during visit
2. livebait : Whether or not group used livebait to fish
3. camper : Whether or not group had a camper
4. persons : Number of adults in group
5. child : Number of children in group
6. hours : Number of hours group spent in park

foxes

Urban foxes

Description

Data on urban fox groups in England.

Usage

```
data(foxes)
```

Format

1. group : ID of group
2. avgfood : Average available food in group's territory
3. groupsize : Size of each group
4. area : Area of group territory
5. weight : Body weight of individual fox

References

Modified from an example in Grafen and Hails.

glimmer	<i>glm/glmer formulas to map/map2stan formulas</i>
---------	--

Description

Converts a glm or glmer model formula into a formula list of the kind used by map and map2stan

Usage

```
glimmer( formula , data , family=gaussian , prefix=c("b_","v_") ,
         default_prior="dnorm(0,10)" , ... )
```

Arguments

formula	A formula of the sort used by glm or glmer
data	A data frame or list containing the data
family	Name of family, as in glm / glmer . Recognizes gaussian, binomial, and poisson with arbitrary links
prefix	Text prefixes for fixed and varying effects parameters
default_prior	Text of default prior for regression parameters
...	Additional parameters to pass to map2stan

Details

This function parses and converts a [glmer](#)-style mixed model formula into a list of formulas appropriate for [map2stan](#) input. Interactions are pre-multiplied, and factors are automatically converted into a series of dummy variables. In the absence of varying effects in the input formula, the resulting formula list will also be suitable for [map](#) input in most cases.

The function [glmer](#) is provided by package [lme4](#), but it is not required for [glimmer](#) to work.

Value

Returns an invisible list with two slots:

f	list of formulas, suitable for passing to map2stan
d	list of data, suitable for passing to map2stan

Author(s)

Richard McElreath

See Also[map2stan](#)**Examples**

```
## Not run:
library(rethinking)
data(UCBadmit)

# varying intercepts
f3 <- cbind(admit,reject) ~ (1|dept) + applicant.gender
m3 <- glimmer( f3 , UCBadmit , binomial )
m3s <- map2stan( m3$f , data=m3$d )

# varying intercepts and slopes
f4 <- cbind(admit,reject) ~ (1+applicant.gender|dept) + applicant.gender
m4 <- glimmer( f4 , UCBadmit , binomial )
m4s <- map2stan( m4$f , data=m4$d )

## End(Not run)
```

HMC2

*Functions for simple HMC simulations***Description**

Conduct simple Hamiltonian Monte Carlo simulations.

Usage

```
HMC2(U, grad_U, epsilon, L, current_q , ... )
HMC_2D_sample( n=100 , U , U_gradient , step , L , start=c(0,0) ,
  xlim=c(-5,5) , ylim=c(-4,4) , xlab="x" , ylab="y" ,
  draw=TRUE , draw_contour=TRUE , nlvls=15 , adj_lvls=1 , ... )
```

Arguments

U	Function to return log density
grad_U	Function to return gradient of U
epsilon	Size of leapfrog step
L	Number of leapfrog steps
current_q	Initial position
n	Number of transitions to sample
step	Size of leapfrog step
start	Initial position
xlim	Horizontal boundaries of plot, for when draw is TRUE

<code>ylim</code>	Vertical boundaries of plot, for when <code>draw</code> is <code>TRUE</code>
<code>draw</code>	Whether to draw the samples and trajectories
<code>draw_contour</code>	Whether to draw contour of density
<code>nlvls</code>	Number of contour levels
<code>adj_lvls</code>	Factor to multiply levels by
<code>...</code>	Optional arguments to pass to density and gradient functions, typically optional parameters

Details

These functions provide simple Hamiltonian Monte Carlo simulations.

HMC2 is based on Neals's 2010 code (see citation below), but returns the trajectories.

HMC_2D_sample simulates multiple sequential trajectories and can also plot the simulation. See examples below.

To use either function, the user must supply custom density and gradient functions.

Author(s)

Richard McElreath

See Also

Radford M. Neal, 2010. MCMC using Hamiltonian dynamics. The Handbook of Markov Chain Monte Carlo.

Examples

```
# Devil's Funnel from Chapter 13
U_funnel <- function( q , s=3 ) {
  v <- q[2]
  x <- q[1]
  U <- sum( dnorm(x,0,exp(v),log=TRUE) ) + dnorm(v,0,s,log=TRUE)
  return( -U )
}
U_funnel_gradient <- function( q , s=3 ) {
  v <- q[2]
  x <- q[1]
  Gv <- (-v)/s^2 - length(x) + exp(-2*v)*sum( x^2 ) #dU/dv
  Gx <- -exp(-2*v)*x #dU/dx
  return( c( -Gx , -Gv ) ) # negative bc energy is neg-log-prob
}
HMC_2D_sample( n=3 , U=U_funnel , U_gradient=U_funnel_gradient ,
  step=0.2 , L=10 , ylab="v" , adj_lvls=1/12 )

# Same, but with non-centered parameterization
U_funnel_NC <- function( q , s=3 ) {
  v <- q[2]
  z <- q[1]
```

```

      U <- sum( dnorm(z,0,1,log=TRUE) ) + dnorm(v,0,s,log=TRUE)
      return( -U )
    }
    U_funnel_NC_gradient <- function( q , s=3 ) {
      v <- q[2]
      z <- q[1]
      Gv <- (-v)/s^2 #dU/dv
      Gz <- (-z) #dU/dz
      return( c( -Gz , -Gv ) ) # negative bc energy is neg-log-prob
    }
    HMC_2D_sample( n=4 , U=U_funnel_NC , U_gradient=U_funnel_NC_gradient ,
      step=0.2 , L=15 , ylab="v" , xlab="z" , xlim=c(-2,2) , nlvls=12 , adj_lvls=1 )

```

Hoogland

Prairie dog dispersal data

Description

Dispersal and kin residence data for three species of prairie dog, from 1976 to 2004. Each row is an individual dispersal record, with associated descriptors.

Usage

```
data(Hoogland)
```

Format

1. Species : 1 = black-tailed, 2 = Gunnison's, 3 = Utah
2. Year : Year of case record
3. Male : 1 = male, 0 = female
4. NoDisperse : 1 = did not disperse within 12 months of weaning, 0 = dispersed
5. Mother : 1 = mother present for 12 months after weaning, 0 = absent
6. Sisters : Minimal number of littermate sisters present at dispersal/non-dispersal
7. Bros : Minimal number of littermate brothers present
8. ClanSize : Minimal number of adults present in territory at time of dispersal/non-dispersal. Includes close kin, distant kin, immigrants, and focal individual
9. AllKin : Minimal number of mother and littermates in territory at time of dispersal/non-dispersal

References

Hoogland 2013. Science 339:1205–1207.

Howell	<i>Howell !Kung demography data</i>
--------	-------------------------------------

Description

Demographic data from Kalahari !Kung San people collected by Nancy Howell

Usage

```
data(Howell1)
data(Howell2)
```

Format

1. height: Height in cm
2. weight: Weight in kg
3. age: Age in years
4. male: Gender indicator
5. age.at.death: If deceased, age at death
6. alive: Indicator if still alive

References

Downloaded from <https://tspace.library.utoronto.ca/handle/1807/10395>

HPDI	<i>Confidence/credible intervals from samples</i>
------	---

Description

These functions compute highest posterior density (HPDI) and percentile (PI) intervals, using samples from a posterior density or simulated outcomes.

Usage

```
HPDI( samples , prob=0.89 )
PI( samples , prob=0.89 )
```

Arguments

samples	Vector of parameter values
prob	interval probability mass

Details

Highest Posterior Density Intervals (HPDI) are calculated by [HPDinterval](#) in the coda package.

Percentile intervals (PI) use [quantile](#) and assign equal mass to each tail.

Author(s)

Richard McElreath

See Also

[HPDinterval](#)

Hurricanes

Hurricane fatalities and gender of names

Description

Data used in Jung et al 2014 analysis of effect of gender of name on hurricane fatalities. Note that hurricanes Katrina (2005) and Audrey (1957) were removed from the data.

Usage

```
data(Hurricanes)
```

Format

1. name : Given name of hurricane
2. year : Year of hurricane
3. deaths : number of deaths
4. category : Severity code for storm
5. min_pressure : Minimum pressure, a measure of storm strength; low is stronger
6. damage_norm : Normalized estimate of damage in dollars
7. female : Indicator variable for female name
8. femininity : 1-11 scale from totally masculine (1) to totally feminine (11) for name. Average of 9 scores from 9 raters.

References

Jung et al. 2014. Female hurricanes are deadlier than male hurricanes. PNAS.

 image_xyz

Heat map from equal length x,y,z vectors

Description

Provides an interface to use `image` by providing three equal length vectors for x, y and z coordinates.

Usage

```
image_xyz( x , y , z , ... )
```

Arguments

x	vector of x values
y	vector of y values
z	vector of z values
...	other parameters to pass to <code>image</code>

Details

This function merely constructs a matrix suitable for `image`, using x, y and z coordinates.

Author(s)

Richard McElreath

See Also

[image](#)

 Kline

Oceanic islands data

Description

Data on historic tool complexity and demography in various Oceanic islands societies. There are three data tables: (1) `Kline` is the basic data table; (2) `Kline2` contains latitude and longitude columns; and (3) `islandsDistMatrix` is a matrix of pairwise distances between islands, in thousands of kilometers.

Usage

```
data(Kline)
data(Kline2)
data(islandsDistMatrix)
```

Format

1. culture : Name of island culture
2. population : Historical population size
3. contact : low or high contact rate with other islands
4. total_tools : number of tools in historical tool kit
5. mean_TU : another measure of tool complexity
6. lat : latitude of island
7. lon : longitude of island
8. lon2 : longitude coded for ease of plotting
9. logpop : natural logarithm of population

References

Kline, M.A. and R. Boyd. 2010. Proc R Soc B 277:2559–2564.

KosterLeckie

Gift exchange data

Description

Data on household gift exchanges from Koster and Leckie. There are two data frames: k1_dyads and k1_households.

Usage

```
data(KosterLeckie)
```

Format

k1_dyads:

1. hidA : household ID for A in dyad
2. hidB : household ID for B in dyad
3. did : dyad ID number
4. giftsAB : count of gifts from A to B
5. giftsBA : count of gifts from B to A
6. offset : relative rate of contact in dyad
7. drel1 :
8. drel2 :
9. drel3 :
10. drel4 :
11. dIndist :

12. dass :

13. d0125 :

kl_households:

1. hid : household ID

2. hgame :

3. hfish :

4. hpigs :

5. hwealth :

6. hpastor :

References

Koster and Leckie (2014) Food sharing networks in lowland Nicaragua: An application of the social relations model to count data. Social Networks.

Laffer

Corporate tax rates and tax revenue for 20 nations

Description

Data on 2004 corporate tax rates and tax revenue for 29 nations, as published in The Wall Street Journal in 2007.

Usage

```
data(Laffer)
```

Format

1. tax_rate: Corporate tax rate
2. tax_revenue: Tax revenue

Author(s)

Richard McElreath

References

"We're Number One, Alas" 2007. The Wall Street Journal.

Description

Computes inverse-link linear model values for map and map2stan samples.

Usage

```
link( fit , data , n=1000 , ... )
## S4 method for signature 'map2stan'
link( fit , data , n=1000 , post , refresh=0.1 ,
      replace=list() , flatten=TRUE , ... )
```

Arguments

fit	Object of class map or map2stan
data	Optional list of data to compute predictions over. When missing, uses data found inside fit object.
n	Number of samples to use
post	Optional samples from posterior. When missing, link extracts the samples using extract.samples .
refresh	Refresh interval for progress display. Set to refresh=0 to suppress display.
replace	Optional named list of samples to replace inside posterior samples. See examples.
flatten	When TRUE, removes linear model names from result
...	Other parameters to pass to someone

Details

This function computes the value of each linear model at each sample for each case in the data. Inverse link functions are applied, so that for example a logit link linear model produces probabilities, using the logistic transform.

This function is used internally by [WAIC](#), [sim](#), [postcheck](#), and [ensemble](#).

It is possible to replace components of the posterior distribution with simulated values. The replace argument should be a named list with replacement values. This is useful for marginalizing over varying effects. See the examples below for an example in which varying intercepts are marginalized this way.

It is easy to substitute zero values for any varying effect parameters in a model. In the data passed to link, either omit the relevant index variable or set the index variable to value zero. This will cause link to replace with zero all samples for any parameters corresponding the index variable in a prior. For example, the prior declaration `aid[id] ~ dmnorm2(0, sigma, Rho)` defines a vector of parameters aid that are indexed by the variable id. If id is absent from data when calling link, or set to value zero, then link will replace all samples for aid with value zero. This effectively removes the varying effect from posterior predictions. If the prior were instead `c(aid, bid)[id] ~ dmnorm(0, sigma, Rho)`, then both aid and bid would be set to zero in all samples.

Author(s)

Richard McElreath

See Also[map](#), [map2stan](#), [sim](#), [ensemble](#), [postcheck](#)**Examples**

```
## Not run:
library(rethinking)
data(chimpanzees)
d <- chimpanzees
d$recipient <- NULL      # get rid of NAs

# model 4 from chapter 12 of the book
m12.4 <- map2stan(
  alist(
    pulled_left ~ dbinom( 1 , p ) ,
    logit(p) <- a + a_actor[actor] + (bp + bpC*condition)*prosoc_left ,
    a_actor[actor] ~ dnorm( 0 , sigma_actor ),
    a ~ dnorm(0,10),
    bp ~ dnorm(0,10),
    bpC ~ dnorm(0,10),
    sigma_actor ~ dcauchy(0,1)
  ) ,
  data=d , warmup=1000 , iter=4000 , chains=4 )

# posterior predictions for a particular actor
chimp <- 2
d.pred <- list(
  prosoc_left = c(0,1,0,1),    # right/left/right/left
  condition = c(0,0,1,1),      # control/control/partner/partner
  actor = rep(chimp,4)
)
link.m12.4 <- link( m12.4 , data=d.pred )
apply( link.m12.4 , 2 , mean )
apply( link.m12.4 , 2 , PI )

# posterior predictions marginal of actor
# here we replace the varying intercepts samples
#   with simulated values

# replace varying intercept samples with simulations
post <- extract.samples(m12.4)
a_actor_sims <- rnorm(7000,0,post$sigma_actor)
a_actor_sims <- matrix(a_actor_sims,1000,7)

# fire up link
# note use of replace list
link.m12.4 <- link( m12.4 , n=1000 , data=d.pred ,
  replace=list(a_actor=a_actor_sims) )
```

```
# summarize
apply( link.m12.4 , 2 , mean )
apply( link.m12.4 , 2 , PI )

## End(Not run)
```

log_sum_exp

Rethinking utility functions

Description

Numerically stable computation of log sums of exponentiated values.

Usage

```
log_sum_exp( x )
```

Arguments

x vector of values

Details

This function is used by WAIC to compute the log average probability used in the formula for WAIC. In that context, a vector of log-probabilities is passed to log_sum_exp, obviating the need to explicitly exponentiate. This helps to avoid rounding errors that occur when working with direct probabilities.

Author(s)

Richard McElreath

map2stan

Build RStan models from formulas

Description

Compiles lists of formulas, like those used in map, into Stan model code. Allows for arbitrary fixed effect and mixed effect regressions. Computes DIC and WAIC. Allows for simple imputation of missing values.

Usage

```
map2stan( flist , data , start , pars , constraints=list() , types=list() ,
  sample=TRUE , iter=2000 , warmup=floor(iter/2) , chains=1 , debug=FALSE ,
  verbose=FALSE , WAIC=TRUE , cores=1 , rng_seed , rawstanfit=FALSE ,
  control=list(adapt_delta=0.95) , add_unique_tag=TRUE , code ,
  log_lik=FALSE , DIC=FALSE , declare_all_data=TRUE ,
  do_discrete_imputation=FALSE , ... )
```

Arguments

<code>flist</code>	A formula or list of formulas that define the likelihood and priors. Can also pass in a map model fit. See details.
<code>data</code>	A data frame or list containing the data
<code>start</code>	Optional named list specifying parameters and their initial values
<code>pars</code>	Optional: character vector of parameters to return samples for
<code>constraints</code>	Optional: named list of custom parameter constraints, using Stan notation
<code>types</code>	Optional: named list of custom parameter types, using Stan notation
<code>sample</code>	If FALSE, builds Stan code without sampling
<code>iter</code>	Number of iterations of sampling. By default, half of these iterations are warmup.
<code>warmup</code>	Number of warmup iterations. By default, half of <code>iter</code> .
<code>chains</code>	Number of independent chains to sample from
<code>debug</code>	If TRUE, prints various internal steps to help with debugging
<code>verbose</code>	If TRUE, prints extra progress messages.
<code>WAIC</code>	When TRUE, computes WAIC after sampling, storing the result
<code>cores</code>	Number of processor cores to distribute chains over, using parallel .
<code>rng_seed</code>	Optional explicit seed.
<code>rawstanfit</code>	When TRUE, returns just the stanfit object, as if the model had been called using stan .
<code>control</code>	Optional list of control parameters for stan. Default increases target acceptance rate (<code>adapt_delta</code>) to 0.95.
<code>add_unique_tag</code>	When TRUE, adds a comment to the Stan model code with the date-time stamp. This makes each model unique and will force Stan to recompile. Useful for avoiding segfault bugs when reusing compiled objects.
<code>code</code>	Optional list of custom Stan code to insert in model. See details and example.
<code>log_lik</code>	Return log likelihood of each observation in samples. Used for calculating WAIC and LOO.
<code>DIC</code>	Return deviance and DIC. This is deprecated and may be removed in a later version.
<code>declare_all_data</code>	When TRUE, all variables in the data list are declared in the Stan model code. When FALSE, only used variables are declared.
<code>do_discrete_imputation</code>	When TRUE, samples for missing binary predictors are returned. Not necessary to marginalize over discrete missing values.
<code>...</code>	Additional arguments to pass to stan

Details

This command provides a convenient interface for building arbitrary fixed effect and mixed effect generalized linear models, as defined by a list of formulas. Syntax is similar to `map`, but also allowing multivariate priors corresponding to varying (aka random) effects, as well as simple imputation schemes.

`flist` should be either (1) a single formula that defines the likelihood function or rather a list of formulas that define the likelihood and linear models and priors for parameters (see examples below) or (2) a previously fit `map` model.

Likelihood formulas take the form $y \sim \text{dfoo}(\text{bar})$, where y is the outcome variable, `dfoo` is a density function such as `dnorm`, and `bar` is a parameter of the density.

Prior formulas take the same form, but the outcome should be a parameter name. Identical priors can be defined for multiple parameters by using `c(par1, par2, ...)` on the left hand side of the formula. See example below.

A special case of prior formula is for varying effects. For single varying effects, such as varying intercepts alone, all that is needed is to define a prior and mark it as conditional on a grouping variable in the data. For example: `aj[id] ~ dnorm(0, sigma_id)` specifies a vector of varying effects `aj`, one for each unique value in `id`. For correlated varying effects, such as both varying intercepts and slopes, a parameter vector is specified and a multivariate prior is used instead. For example: `c(aj, bj)[id] ~ dmnorm(0, Sigma_id)` specifies varying intercepts `aj` and varying slopes `bj`.

Linear models can be specified as formulas of the form $\mu \leftarrow a + b \cdot x$ for a direct link. To use a link function, use the form `link(mu) <- a + b * x`. The name "link" must be recognized by `map2stan`. It currently recognizes `log` and `logit`.

Imputation of missing values is available by specifying distributions for predictor variables that contain NA values. `map2stan` will split the variable into observed values and a vector of parameters used to estimate the missing values, assigning the same distribution to each. See the example.

When predictor variables are binary (0/1), `map2stan` will attempt to marginalize over any missing values. This is accomplished by building a mixture likelihood. Missingness in more than one binary variable can be accommodated this way, by building a list of all combinations of missingness among the variables and then a correspond vector of likelihood terms. The resulting Stan code contains a loop that computes the proper mixture and adds it to the target with `log_sum_exp`. The user may need to use the optional `constraints` list to bound hyperparameters. See the example.

The `start` list is optional. When missing from the list, for each parameter with a defined prior, an initial value will be sampled from the prior. Sampled initial values will try to respect parameter constraints. For varying effect parameter vectors, initial values will always be set to zero. Specific initial values can be specified in the `start` list. See examples below.

The optional `code` argument can be used to pass a list of custom Stan code to be inserted into specific parts of the model. The list should be a list of lists. Each list should have the format `list("code", block="block", section="section", pos="pos")`. The first argument is the code to insert, as a character string. The named `block` slot should be one of `functions`, `data`, `transformed data`, `parameters`, `transformed parameters`, `model`, or `generated quantities`. The named `section` slot should be one of `declare` or `body`, specifying whether the new code appears in the declared variables header or rather the code body of a block. The named `pos` slot should be one of `top`, `bottom`, or `pattern`. The position `pattern` uses the additional named slot `pattern` to search-and-replace, replacing the text in `pattern` with the text in the first argument. See the example at the end of this help page.

The Stan model code includes a generated quantities block that computes the deviance for each iteration of parameter samples. When sampling completes, map2stan computes DIC, the deviance information criterion, from the samples. DIC information is available from show and DIC, as well as being attributes of the returned object.

WAIC can be computed with WAIC, or by setting WAIC=TRUE when calling map2stan. This is currently the default. WAIC is calculated entirely after Stan completes sampling.

Methods are defined for [extract.samples](#), [link](#), [sim](#), [ensemble](#), [compare](#), [coef](#), [summary](#), [logLik](#), [vcov](#), [nobs](#), [deviance](#), [plot](#), [pairs](#), and [show](#).

Value

Returns an object of class map2stan with the following slots.

call	The function call
model	Stan model code
stanfit	stanfit object returned by stan
coef	The posterior means
vcov	Minimal variance-covariance matrix, just holding diagonal variances
data	The data
start	List of starting values that were used in sampling
pars	Parameter names monitored in samples
formula	Formula list from call
formula_parsed	List of parsed formula information. Useful mainly for debugging.

Author(s)

Richard McElreath

See Also

[resample](#), [map](#), [stan](#), [link](#), [sim](#), [glimmer](#)

Examples

```
## Not run:
library(rethinking)
data(chimpanzees)

# don't want any variables with NAs
d <- list(
  pulled_left = chimpanzees$pulled_left ,
  prosoc_left = chimpanzees$prosoc_left ,
  condition = chimpanzees$condition ,
  actor = as.integer( chimpanzees$actor ) ,
  blockid = as.integer( chimpanzees$block )
)
```

```

# RStan fit
m2 <- map2stan(
  alist(
    pulled_left ~ dbinom(1,theta),
    logit(theta) <- a + bp*prosoc_left + bpc*condition*prosoc_left ,
    a ~ dnorm(0,10),
    bp ~ dnorm(0,10),
    bpc ~ dnorm(0,10)
  ) ,
  data=d, chains=2, cores=1 )

precis(m2)
summary(m2)
plot(m2)
pairs(m2)

# now RStan fit of model with varying intercepts on actor
m3 <- map2stan(
  alist(
    pulled_left ~ dbinom(1,theta),
    logit(theta) <- a + aj[actor] + bp*prosoc_left + bpc*condition*prosoc_left,
    aj[actor] ~ dnorm( 0 , sigma_actor ),
    a ~ dnorm(0,10),
    bp ~ dnorm(0,10),
    bpc ~ dnorm(0,10),
    sigma_actor ~ dcauchy(0,1)
  ) ,
  data=d,
  iter=5000 , warmup=1000 , chains=2 , cores=1 )

precis(m3)
plot(m3)
pairs(m3)

# varying intercepts on actor and experimental block
m4 <- map2stan(
  alist(
    pulled_left ~ dbinom(1,theta),
    logit(theta) <- a + aj + ak + bp*prosoc_left + bpc*condition*prosoc_left,
    aj[actor] ~ dnorm( 0 , sigma_actor ),
    ak[blockid] ~ dnorm( 0 , sigma_block ),
    a ~ dnorm(0,10),
    bp ~ dnorm(0,10),
    bpc ~ dnorm(0,10),
    sigma_actor ~ dcauchy(0,1),
    sigma_block ~ dcauchy(0,1)
  ) ,
  data=d,
  iter=5000 , warmup=1000 , chains=2 , cores=1 )

precis(m4)
summary(m4)
plot(m4)

```

```

# compare posterior means
coefstab(m2,m3,m4)
plot(coefstab(m2,m3,m4))

# show WAIC for m2,m3,m4
compare(m2,m3,m4)
plot(compare(m2,m3,m4))

#####
# varying slopes models

# varying slopes on actor
# also demonstrates use of multiple linear models
# see Chapter 13 for discussion
m5 <- map2stan(
  alist(
    # likelihood
    pulled_left ~ dbinom(1,p),

    # linear models
    logit(p) <- A + (BP + BPC*condition)*prosoc_left,
    A <- a + a_actor[actor],
    BP <- bp + bp_actor[actor],
    BPC <- bpc + bpc_actor[actor],

    # adaptive prior
    c(a_actor,bp_actor,bpc_actor)[actor] ~
      dmvmnorm2(0,sigma_actor,Rho_actor),

    # fixed priors
    c(a,bp,bpc) ~ dnorm(0,1),
    sigma_actor ~ dcauchy(0,2),
    Rho_actor ~ dlkjcorr(4)
  ) , data=d , iter=5000 , warmup=1000 , chains=3 , cores=3 )

# same model but with non-centered parameterization
# see Chapter 13 for explanation and more elaborate example

m6 <- map2stan(
  alist(
    # likelihood
    pulled_left ~ dbinom(1,p),

    # linear models
    logit(p) <- A + (BP + BPC*condition)*prosoc_left,
    A <- a + a_actor[actor],
    BP <- bp + bp_actor[actor],
    BPC <- bpc + bpc_actor[actor],

    # adaptive prior - non-centered
    c(a_actor,bp_actor,bpc_actor)[actor] ~
      dmvmnormNC(sigma_actor,Rho_actor),

```

```

      # fixed priors
      c(a,bp,bpc) ~ dnorm(0,1),
      sigma_actor ~ dcauchy(0,2),
      Rho_actor ~ dlkjcorr(4)
    ) , data=d , iter=5000 , warmup=1000 , chains=3 , cores=3 )

#####
# Imputation example

# simulate data:
# linear regression with two predictors
# both predictors have valules missing at random
N <- 100
N_miss <- 10
x1 <- rnorm( N )
x2 <- rnorm( N )
y <- rnorm( N , 2*x1 - 0.5*x2 , 1 )
x1[ sample(1:N,size=N_miss) ] <- NA
x2[ sample(1:N,size=N_miss) ] <- NA

# formula with distributions assigned to both predictors
f <- alist(
  y ~ dnorm( mu , sigma ),
  mu <- a + b1*x1 + b2*x2,
  x1 ~ dnorm( mu_x1, sigma_x1 ),
  x2 ~ dnorm( mu_x2, sigma_x2 ),
  a ~ dnorm( 0 , 100 ),
  c(b1,b2) ~ dnorm( 0 , 10 ),
  c(mu_x1,mu_x2) ~ dnorm( 0 , 100 ),
  c(sigma_x1,sigma_x2) ~ dcauchy(0,2),
  sigma ~ dcauchy(0,2)
)

m <- map2stan( f , data=list(y=y,x1=x1,x2=x2) , sample=TRUE )

# show observed outcomes against retrodicted outcomes
# cases with missing values shown with red posterior intervals
v <- link(m)
mu <- apply( v , 2 , mean )
ci <- apply( v , 2 , PI )
plot( y ~ mu )
cicols <- ifelse( is.na(x1) | is.na(x2) , "red" , "gray" )
for( i in 1:N ) lines( ci[,i] , rep(y[i],2) , col=cicols[i] )

#####
# Binary marginalization example

# Simulate data
N <- 100
N_miss <- 10
x1 <- rbinom( N , 1 , 0.5 )
x2 <- rbinom( N , 1 , 0.2 )

```

```

y <- rnorm( N , 2*x1 - 0.5*x2 , 1 )
x1[ sample(1:N,size=N_miss) ] <- NA
x2[ sample(1:N,size=N_miss) ] <- NA

# Formula with distributions assigned to both predictors
f <- alist(
  y ~ dnorm( mu , sigma ),
  mu <- a + b1*x1 + b2*x2,
  x1 ~ bernoulli( phi_x1 ),
  x2 ~ bernoulli( phi_x2 ),
  a ~ dnorm( 0 , 100 ),
  c(b1,b2) ~ dnorm( 0 , 10 ),
  c(phi_x1,phi_x2) ~ beta( 2 , 2 ),
  sigma ~ dcauchy(0,2)
)

m <- map2stan( f , data=list(y=y,x1=x1,x2=x2) ,
  constraints=list(phi_x1="lower=0,upper=1",phi_x2="lower=0,upper=1") )

# Inspect model block of the Stan code to see how the mixture is built.
stancode(m)

# Note that the matrix mu_missmatrix is passed as data and contains the combinations of missingness. Columns are var
m@data$mu_missmatrix

#####
# custom code insertion

N <- 1000
y <- rnorm( N )

m <- map2stan(
  alist(
    y ~ normal(mu,sigma),
    mu <- a,
    a ~ normal(0,10),
    sigma ~ exponential(1)
  ),
  data=list(y=y),
  code=list(
    list("//test",block="data",pos="top"),
    list("//test2",block="parameters",pos="bottom"),
    list("//test3",block="model",section="declare",pos="bottom"),
    list("--test4--",block="model",section="declare",pos="pattern",pattern="test3"),
    list("real asq;",block="transformed parameters",section="declare"),
    list("asq = a*a;",block="transformed parameters",section="body")
  ),
  sample=FALSE )

stancode(m)

## End(Not run)

```

map2stan-class	<i>Class map2stan: fitted map2stan Stan model</i>
----------------	---

Description

This object contains an object of class `stanfit`, return by [stan](#), along with other information produced by a [map2stan](#) model fit.

Slots

`call`: Original function call that produced the fit
`model`: Text of the Stan model used in sampling
`stanfit`: Object of class `stanfit`
`coef`: Vector of posterior means
`vcov`: Diagonal matrix with parameter variances
`data`: Data used in sampling
`start`: Initial values used in sampling
`pars`: Vector of parameters returned by sampling
`formula`: Original `alist` of formulas used to define model
`formula_parsed`: Parsed formula list. Used during compilation and useful for debugging and class methods.

Methods

`show signature(object = "map2stan")`: print the default summary for the model.
`stancode` Displays the model slot.
`WAIC` Computes WAIC

See Also

[map2stan](#)

mcreplicate	<i>Multi-core version of replicate</i>
-------------	--

Description

Uses the parallel library to distribute [replicate](#) processing across cores.

Usage

```
mcreplicate(n, expr, refresh = 0.1, mc.cores=2 )
```

Arguments

n	Number of replications
expr	Code to replicate
refresh	Status update refresh interval
mc.cores	Number of cores to use

Details

This function uses [mclapply](#) to distribute replications across cores. It then simplifies the result to an array.

Author(s)

Richard McElreath

See Also

[mclapply](#), [replicate](#)

milk

Primate milk data

Description

Comparative primate milk composition data, from Table 2 of Hinde and Milligan. 2011. *Evolutionary Anthropology* 20:9-23.

Usage

```
data(milk)
```

Format

Returns a data frame containing 29 rows and 8 columns.

1. clade: Broad taxonomic group
2. species: Species name
3. kcal.per.g: Kilocalories per gram of milk
4. perc.fat: Percent fat
5. perc.protein: Percent protein
6. perc.lactose: Percent lactose
7. mass: Body mass of mother, in kilograms
8. neocortex.perc: Percent of brain mass that is neocortex

Author(s)

Richard McElreath

References

Hinde and Milligan. 2011. Evolutionary Anthropology 20:9-23.

Moralizing_gods

Religious ideology and population growth

Description

Historical data on human societies, their population sizes, and the present or absence of moralizing gods. Moralizing gods are defined as gods that are believed to enforce prosocial behavior through punishment or reward.

Usage

```
data(Moralizing_gods)
```

Format

1. polity: Name of society
2. year: Calendar year of record
3. population: Log population of polity
4. moralizing_gods: Indicator of presence (1), absence (0), or unknown (NA)
5. writing: Presence of written records

References

Whitehouse et al. 2019. Complex societies precede moralizing gods throughout world history. doi:10.1038/s41586-019-1043-4

pairs.map2stan	<i>Trace plot of map2stan chains</i>
----------------	--------------------------------------

Description

Shows pairs plots for Stan samples produced by a map or map2stan fit.

Usage

```
pairs(x, n=500 , alpha=0.7 , cex=0.7 , pch=16 , adj=1 , pars , ...)
```

Arguments

x	map or map2stan model fit
n	Number of samples to show in scatter plots
alpha	alpha transparency of scatter plots
cex	Character expansion factor for scatter plots
pch	Point type for scatter plots
adj	adj argument for density estimate
pars	Character vector of parameters to display
...	Additional plot arguments

Details

This is the default pairs plot method for both [map](#) and [map2stan](#) model fits.

Author(s)

Richard McElreath

Panda_nuts	<i>Chimpanzees cracking Panda nuts</i>
------------	--

Description

Data on 22 western chimpanzees attempting to crack Panda nuts.

Usage

```
data(Panda_nuts)
```

Format

1. chimpanzee: ID of individual
2. age: Age in years at time of observation
3. sex: individual's sex
4. hammer: Type of hammer (one of four types)
5. nuts_opened: Number of nuts opened in session
6. seconds: Duration of session in seconds
7. help: Received help from another chimpanzee (yes or no)

References

Boesch, Bombjakova, Meier, and Mundry. 2019. "Learning curves and teaching when acquiring nut-cracking in humans and chimpanzees". doi:10.1038/s41598-018-38392-8

plot.map2stan	<i>Trace plot of map2stan chains</i>
---------------	--------------------------------------

Description

Shows trace plots for Stan samples produced by a map2stan fit, annotated by number of effective samples. Automatic paging and adjustable number of columns and rows per page.

Usage

```
plot( object , pars , col=rethink_palette , alpha=1 , bg=gray(0.7,0.5) , ask=TRUE , window , n_cols=3 , m
```

Arguments

object	map2stan model fit
pars	Character vector of parameters to display
col	Vector of colors to use for chains. Recycled.
alpha	alpha transparency of chains
bg	Background fill for warmup samples
ask	Whether to pause for paging. Default is TRUE.
window	Range of samples to display. Default is all samples.
n_cols	Number of columns per page
max_rows	Maximum number of rows per page
...	Additional arguments to pass to plot commands

Details

This is the default trace plot method for [map2stan](#) model fits.

Author(s)

Richard McElreath

postcheck	<i>Plot posterior predictive check</i>
-----------	--

Description

Plots a series of graphical posterior predictive checks for a map or map2stan model fit.

Usage

```
postcheck( fit , prob=0.9 , window=20 , n=1000 , col=rangi2 , ... )
```

Arguments

fit	A map or map2stan model fit
prob	Width of prediction interval to display
window	Number of cases to display in each plot (for paging)
n	Number of samples from posterior to use
...	Additional arguments to pass to plot

Details

This function uses [link](#) and [sim](#) internally to simulate posterior predictions for a fit model. These are then plotted over the observed outcomes used in fitting.

Author(s)

Richard McElreath

See Also

[link](#), [sim](#)

precis	<i>Precis of model fit</i>
--------	----------------------------

Description

Displays concise parameter estimate information for an existing model fit.

Usage

```
precis( model , depth=1 , pars , ci=TRUE , prob=0.89 ,  
        corr=FALSE , digits=2 , warn=TRUE )
```

Arguments

<code>model</code>	Fit model object
<code>depth</code>	If 1, suppresses vectors and matrices of parameters. If 2, displays all parameters
<code>pars</code>	Optional character vector of parameter names to display
<code>ci</code>	Show quadratic estimate confidence intervals
<code>prob</code>	Width of posterior intervals
<code>corr</code>	If TRUE, show correlations among parameters in output
<code>digits</code>	Number of decimal places to display in output
<code>warn</code>	If TRUE, warns about various things

Details

Creates a table of estimates and standard errors, with optional confidence intervals and parameter correlations. Posterior intervals are quadratic estimates, derived from standard deviations, unless the model uses samples from the posterior distribution, in which case [HPDI](#) is used instead.

Can also provide expected value, standard deviation, and HPDI columns for a data frame.

Value

A data frame with a row for each parameter. The `n_eff` and `Rhat` columns are calculated by `rstan`. `Rhat4` indicates `Rhat` as defined in Gelman et al 2013 "Bayesian Data Analysis". This is different from the classic 1992 Gelman and Rubin definition, as `Rhat4` uses more information from multiple chains.

Author(s)

Richard McElreath

Primates301

Primate life history and social learning data

Description

Life history data, social learning data, and phylogenetic distance matrix for 301 primate species. These data were assembled by and analyzed in Street et al 2017 (see references).

Usage

```
data(Primates301)
data(Primates301_distance_matrix)
data(Primates301_vcov_matrix)
```

Format

Primates301 is a data.table with elements:

1. name: Full taxonomic name of species
2. genus : Genus of species
3. species : Species name within genus
4. subspecies : Sub-species designation, if any
5. spp_id : Unique ID for species
6. genus_id : Unique ID for genus
7. social_learning : Count of mentions of social learning in literature
8. research_effort : Size of literature on species
9. brain : Brain volume (endocranial volume) in cubic centimeters
10. body : Body mass in grams
11. group_size : Average social group size
12. gestation : Length of gestation (days)
13. weaning : At at weaning (days)
14. longevity : Maximum lifespan (months)
15. sex_maturity : Age of sexual maturity (days)
16. maternal_investment : Period of maternal investment (days) = gestation + weaning

Primates301_distance_matrix is a matrix with species on the margins and phylogenetic distances in the cells. Primates301_vcov_matrix is a matrix with species on the margins and variances-covariances in the cells.

References

- Street SE, Navarrete AF, Reader SM, Laland KN (2017) Coevolution of cultural intelligence, extended life history, sociality, and brain size in primates. PNAS <https://doi.org/10.1073/pnas.1620734114>
- Arnold C, Matthews LJ, Nunn CL (2010) The 10kTrees Website: A New Online Resource for Primate Phylogeny. *Evol Anthropol* 19(3):114-118.
- Reader SM, Hager Y, Laland KN (2011) The evolution of primate general and cultural intelligence. *Philos Trans R Soc B-Biological Sci* 366(1567):1017-1027.
- Isler K, et al. (2008) Endocranial volumes of primate species: scaling analyses using a comprehensive and reliable data set. *J Hum Evol* 55(6):967-978.
- Jones, Kate E, et al. (2009) PanTHERIA: a species-level database of life history, ecology, and geography of extant and recently extinct mammals. *Ecology* 90:2649.

Examples

```
data(Primates301)
plot( log(brain) ~ log(body) , data=Primates301 )

data(Primates301_distance_matrix)
image(Primates301_distance_matrix)
```

```

# Gaussian process phylogenetic regression
# prep variables
d <- Primates301
d$name <- as.character(d$name)
dstan <- d[ complete.cases( d$social_learning, d$research_effort , d$body , d$brain ) , ]
# prune distance matrix to spp in dstan
spp_obs <- dstan$name
y <- Primates301_distance_matrix
y2 <- y[ spp_obs , spp_obs ]
# cbind( sort(spp_obs) , sort(colnames(y2)) )
# scale distances
y3 <- y2/max(y2)

mP301GP <- ulam(
  alist(
    social_learning ~ poisson( lambda ),
    log(lambda) <- a + g[spp_id] + b_ef*log_research_effort + b_body*log_body + b_eq*log_brain,
    a ~ normal(0,1),
    vector[N_spp]: g ~ multi_normal( 0 , SIGMA ),
    matrix[N_spp,N_spp]: SIGMA <- cov_GPL2( Dmat , etasq , rhosq , 0.01 ),
    b_body ~ normal(0,1),
    b_eq ~ normal(0,1),
    b_ef ~ normal(1,1),
    etasq ~ exponential(1),
    rhosq ~ exponential(1)
  ),
  data=list(
    N_spp = nrow(dstan),
    social_learning = dstan$social_learning,
    spp_id = 1:nrow(dstan),
    log_research_effort = log(dstan$research_effort),
    log_body = log(dstan$body),
    log_brain = log(dstan$brain),
    Dmat = y3
  ) ,
  control=list(max_treedepth=15,adapt_delta=0.95) ,
  sample=FALSE , iter=400 )

# non-centered, Cholesky form
mP301GPnc <- ulam(
  alist(
    social_learning ~ poisson( lambda ),
    log(lambda) <- a + g[spp_id] + b_ef*log_research_effort + b_body*log_body + b_eq*log_brain,
    a ~ normal(0,1),
    vector[N_spp]: g <- L_SIGMA * eta,
    vector[N_spp]: eta ~ normal( 0 , 1 ),
    matrix[N_spp,N_spp]: L_SIGMA <- cholesky_decompose( SIGMA ),
    matrix[N_spp,N_spp]: SIGMA <- cov_GPL2( Dmat , etasq , rhosq , 0.01 ),
    b_body ~ normal(0,1),
    b_eq ~ normal(0,1),
    b_ef ~ normal(1,1),
    etasq ~ exponential(1),

```

```

      rhosq ~ exponential(1)
    ),
    data=list(
      N_spp = nrow(dstan),
      social_learning = dstan$social_learning,
      spp_id = 1:nrow(dstan),
      log_research_effort = log(dstan$research_effort),
      log_body = log(dstan$body),
      log_brain = log(dstan$brain),
      Dmat = y3
    ),
    control=list(max_treedepth=15,adapt_delta=0.95) ,
    sample=FALSE , iter=400 )

# Pagel's lambda approach --- Not endorsed!
# This is of historical interest only
data(Primates301_vcov_matrix)
vcov_thin <- Primates301_vcov_matrix[ spp_obs , spp_obs ]
mP301L <- ulam(
  alist(
    social_learning ~ poisson( lambda ),
    log(lambda) <- a + g[spp_id] + b_ef*log_research_effort + b_body*log_body + b_eq*log_brain,
    a ~ normal(0,1),
    vector[N_spp]: g <- L_SIGMA * eta,
    vector[N_spp]: eta ~ normal( 0 , 1 ),
    matrix[N_spp,N_spp]: L_SIGMA <- cholesky_decompose( SIGMA ),
    matrix[N_spp,N_spp]: SIGMA <- cov_Pagel( SIGMA_raw , Plambda ),
    b_body ~ normal(0,1),
    b_eq ~ normal(0,1),
    b_ef ~ normal(1,1),
    Plambda ~ beta(2,2)
  ),
  data=list(
    N_spp = nrow(dstan),
    social_learning = dstan$social_learning,
    spp_id = 1:nrow(dstan),
    log_research_effort = log(dstan$research_effort),
    log_body = log(dstan$body),
    log_brain = log(dstan$brain),
    SIGMA_raw = vcov_thin
  ),
  control=list(max_treedepth=15,adapt_delta=0.95) ,
  sample=TRUE , iter=400 )

# centered version --- seems to mix better
mP301L2 <- ulam(
  alist(
    social_learning ~ poisson( lambda ),
    log(lambda) <- a + g[spp_id] + b_ef*log_research_effort + b_body*log_body + b_eq*log_brain,
    a ~ normal(0,1),
    vector[N_spp]: g ~ multi_normal( 0 , SIGMA ),
    matrix[N_spp,N_spp]: SIGMA <- cov_Pagel( SIGMA_raw , Plambda ),
    b_body ~ normal(0,1),

```

```

      b_eq ~ normal(0,1),
      b_ef ~ normal(1,1),
      Plambda ~ beta(2,2)
    ),
    data=list(
      N_spp = nrow(dstan),
      social_learning = dstan$social_learning,
      spp_id = 1:nrow(dstan),
      log_research_effort = log(dstan$research_effort),
      log_body = log(dstan$body),
      log_brain = log(dstan$brain),
      SIGMA_raw = vcov_thin
    ) ,
    control=list(max_treedepth=15,adapt_delta=0.95) ,
    sample=TRUE , iter=400 )

```

progbar

Progress display

Description

Provides a progress display with estimated time remaining, assuming rate constant process.

Usage

```

progbar(current, min = 0, max = 100, starttime,
        update.interval = 100, show.rate = FALSE)

```

Arguments

current	Current loop index of process
min	Minimum loop index, usually 0 or 1
max	Maximum loop index
starttime	Time stamp when process began, from Sys.time
update.interval	How often to display progress

Details

This function provides useful progress information and estimated time until completion for long looped operations, like sampling from MCMC.

Author(s)

Richard McElreath

PrussianHorses	<i>Numbers of soldiers killed by horse kicks in the Prussian army 1875-1894</i>
----------------	---

Description

Data collected by Ladsilaus von Bortkiewicz (1898), Das Gesetz der kleinen Zahlen.

Usage

```
data(PrussianHorses)
```

Format

1. kicks: Count of soldier deaths from horse kicks
2. year: Calendar year
3. corps: Military corps (division). G is the guard corps.

Author(s)

Richard McElreath

References

von Bortkiewicz (1898), Das Gesetz der kleinen Zahlen

Examples

```
data(PrussianHorses)
d <- PrussianHorses

dat <- list(
  kicks = d$kicks,
  year = as.integer(as.factor(d$year)), # year as category
  corps = as.integer(d$corps)
)

m0 <- ulam(
  alist(
    kicks ~ poisson( lambda ),
    log(lambda) <- d[corps] + y[year],
    d[corps] ~ normal(0,0.5),
    y[year] ~ normal(0,0.5)
  ), data=dat )

plot(precis(m0,2))
```

quap

*Compute quadratic approximate posterior distribution***Description**

Find mode of posterior distribution for arbitrary fixed effect models and then produce an approximation of the full posterior using the quadratic curvature at the mode.

Usage

```
quap( flist , data , start , method="BFGS" , hessian=TRUE , debug=FALSE , verbose=FALSE , ... )
map( flist , data , start , method="BFGS" , hessian=TRUE , debug=FALSE , verbose=FALSE , ... )
```

Arguments

<code>flist</code>	A formula or alist of formulas that define the likelihood and priors. See details.
<code>data</code>	A data frame or list containing the data
<code>start</code>	Optional named list specifying parameters and their initial values
<code>method</code>	Search method for <code>optim</code> . Defaults to BFGS.
<code>hessian</code>	If TRUE, attempts to compute the Hessian
<code>debug</code>	If TRUE, prints various internal steps to help with debugging
<code>...</code>	Additional arguments to pass to <code>optim</code>

Details

This command provides a convenient interface for finding quadratic approximations of posterior distributions for models defined by a formula or by a list of formulas. This procedure is equivalent to penalized maximum likelihood estimation and the use of a Hessian for profiling, and therefore can be used to define many common regularization procedures. The point estimates returned correspond to a maximum a posterior, or MAP, estimate. However the intention is that users will use `extract.samples` and other functions to work with the full posterior.

`flist` should be either a single formula that defines the likelihood function or rather a list of formulas that define the likelihood and priors for parameters. See examples below.

Likelihood formulas take the form $y \sim \text{dfoo}(\text{bar})$, where y is the outcome variable, `dfoo` is a density function such as `dnorm`, and `bar` is a parameter of the density.

Prior formulas take the same form, but the outcome should be a parameter name. Identical priors can be defined for multiple parameters by using `c(par1, par2, ...)` on the left hand side of the formula. See example below.

Linear models can be specified as formulas of the form $\mu \leftarrow a + b \cdot x$ for a direct link. To use a link function, use the form `link(mu) <- a + b * x` or `mu <- invlink(a + b * x)`. The names "link" and "invlink" must be recognized by `map`. It currently recognizes log/exp and logit/logistic. Any other link function can be coded directly into the likelihood formula. For example $y \sim \text{dfoo}(\text{invlink}(\mu))$.

The `start` list is optional. For each parameter with a defined prior, an initial value will be sampled from the prior. Explicit named initial values can also be provided in this list.

Methods are defined for `coef`, `summary`, `logLik`, `vcov`, `nobs`, `deviance`, `link`, [DIC](#), and `show`.

Value

Returns an object of class `map` with the following slots.

<code>call</code>	The function call
<code>coef</code>	The estimated posterior modes
<code>vcov</code>	Variance-covariance matrix
<code>optim</code>	List returned by <code>optim</code>
<code>data</code>	The data
<code>formula</code>	Formula list
<code>fminuslogl</code>	Minus log-likelihood function that accepts a vector of parameter values

Author(s)

Richard McElreath

See Also

[optim](#)

Examples

```
data(cars)
flist <- alist(
  dist ~ dnorm( mu , sigma ) ,
  mu <- a+b*speed ,
  c(a,b) ~ dnorm(0,1) ,
  sigma ~ dexp(1)
)
fit <- quap( flist , start=list(a=40,b=0.1,sigma=20) , data=cars )

# regularized logistic regression example
y <- c( rep(0,10) , rep(1,10) )
x <- c( rep(-1,9) , rep(1,11) )

flist0 <- alist(
  y ~ dbinom( 1 , p ) ,
  logit(p) <- a + b*x
)

flist1 <- alist(
  y ~ dbinom( 1 , p ),
  logit(p) <- a + b*x ,
  c(a,b) ~ dnorm(0,10)
)

# without priors, same problem as:
# glm3a <- glm( y ~ x , family=binomial , data=list(y=y,x=x) )
fit3a <- quap( flist0 , data=list(y=y,x=x) , start=list(a=0,b=0) )
precis(fit3a)
```

```
# now with regularization
fit3b <- quap( flist1 , data=list(y=y,x=x) , start=list(a=0,b=0) )
precis(fit3b)

# vector parameters
data(UCBadmit)
fit4 <- quap(
  alist(
    admit ~ dbinom( apps , p ),
    logit(p) <- a[dept_id] + b*male,
    a[dept_id] ~ dnorm(0,4),
    b ~ dnorm(0,1)
  ),
  data=list(
    admit = UCBadmit$admit,
    apps = UCBadmit$applications,
    male = ifelse( UCBadmit$applicant.gender=="male" , 1 , 0 ),
    dept_id = coerce_index(UCBadmit$dept)
  )
)
```

reedfrogs

Data on reed frog predation experiments

Description

Data on lab experiments on the density- and size-dependent predation rate of an African reed frog, *Hyperolius spinigularis*, from Vonesh and Bolker 2005

Usage

```
data(reedfrogs)
```

Format

Various data with variables:

density initial tadpole density (number of tadpoles in a 1.2 x 0.8 x 0.4 m tank) [experiment 1]

pred factor: predators present or absent [experiment 1]

size factor: big or small tadpoles [experiment 1]

surv number surviving

propsurv proportion surviving (=surv/density) [experiment 1]

Source

Vonesh and Bolker (2005) Compensatory larval responses shift trade-offs associated with predator-induced hatching plasticity. *Ecology* 86:1580-1591

Examples

```
data(reedfrogs)
boxplot(propsurv~size*density*pred,data=reedfrogs)
```

resample

*Resample map2stan fit***Description**

Sample from a new chain or chains, using a previous map2stan fit object.

Usage

```
resample( object , iter=1e4 , warmup=1000 , chains=1 , cores=1 ,
          DIC=TRUE , WAIC=TRUE , rng_seed , data , ... )
```

Arguments

object	Object of class map2stan
iter	Number of sampling iterations, including warmup
warmup	Number of adaptation steps
chains	Number of independent chains
cores	Number of cores to distribute chains across
DIC	If TRUE, computes DIC after sampling
WAIC	If TRUE, computes WAIC after sampling
rng_seed	Optional seed to use for all chains. When missing, a random seed is chosen and used for all chains.
...	Other parameters to pass to stan

Details

This function is a convenience for drawing more samples from an initial map2stan fit.

When cores is set greater than 1, either [mclapply](#) (on a unix system) or [parLapply](#) (on a Windows system) is used to run the chains, distributing them across processor cores. The results are automatically recombined with [sflist2stanfit](#).

Value

An object of class map2stan, holding the new samples, as well as all of the original formulas and data for the model.

Author(s)

Richard McElreath

See Also

[map2stan](#), [mclapply](#), [sflist2stanfit](#)

Examples

```
## Not run:
data(Trolley)
d <- Trolley
d2 <- list(
  y=d$response,
  xA=d$action,
  xI=d$intention,
  xC=d$contact,
  id=as.integer(d$id)
)
Nid <- length(unique(d2$id))

# ordered logit regression with varying intercepts
m.init <- map2stan(
  alist(
    y ~ dordlogit( phi , cutpoints ),
    phi <- aj + bA*xA + bI*xI + bC*xC,
    c(bA,bI,bC) ~ dnorm(0,1),
    aj[id] ~ dnorm(0,sigma_id),
    sigma_id ~ dcauchy(0,2.5),
    cutpoints ~ dcauchy(0,2.5)
  ),
  data=d2 ,
  start=list(
    bA=0,bI=0,bC=0,
    cutpoints=c(-2,-1.7,-1,-0.2,0.5,1.3),
    aj=rep(0,Nid),sigma_id=1
  ),
  types=list(cutpoints="ordered") ,
  iter=2
)

# Note: parallel chains won't work on Windows
m <- resample( m.init , chains=3 , cores=3 , warmup=1000 , iter=3000 )

## End(Not run)
```

Description

Mortality counts for 11 herds of large and small livestock. Large livestock are Zebu cattle, and small livestock are a mix of (fat-tailed, Maasai) sheep and goats.

Usage

```
data(Rinder)
```

Format

1. herd : identifier for individual herd
2. stock : categorical small or large stock
3. n : number of animals at beginning of observation period
4. mortality : number of animals that died in observation period

<code>sample.qa.posterior</code>	<i>Samples from quadratic posterior densities of models</i>
----------------------------------	---

Description

Samples from the posterior density of a fit model or models, assuming multivariate normal density.

Usage

```
sample.qa.posterior(model, n = 10000, clean.names = TRUE, model.weights =
  "AICc", nobs = 0, add.names = FALSE, fill.na = 0,
  verbose = FALSE)
```

Arguments

<code>model</code>	A fit model object
<code>models</code>	A list of fit models of the same class
<code>n</code>	Number of samples to draw from joint posterior
<code>model.weights</code>	If passing a list of models, method for computing posterior probability of each model family. Can be "AIC", "AICc", "BIC" or a vector of numeric weights.
<code>nobs</code>	Number of observations used to fit model or all models in list. Sometimes needed for <code>model.weights</code> values, like AICc.
<code>add.names</code>	Adds a column of model names, when passing a list of models
<code>fill.na</code>	Fills missing values with 0, by default, for model families that do not contain a given parameter. Useful for linear models. Hazardous for non-linear ones.
<code>verbose</code>	If TRUE, prints various debugging information

Details

This is a legacy function and is no longer supported nor unit tested.

This function provides a way to draw parameter values from a multivariate normal posterior density, estimated from the maximum a posteriori (MAP) estimates and variance-covariance (vcov) of a fit model or models.

When passing a single fit model object, the function returns a data frame in which each row is a sample and each column is a parameter.

When passing a list of fit model objects, the function returns a data frame containing samples from the joint posterior across model families. The fraction of rows drawn from a specific model family is determined by the `model.weights` parameter. BIC, AIC, or AICc are used to compute approximate predictive probabilities of each model family, and the total samples `n` is proportioned according to these estimates. The user can also supply a numeric vector of model weights, computed by any method. This vector should sum to 1.

Author(s)

Richard McElreath

See Also

[mvrnorm](#)

shade	<i>Shade density intervals</i>
-------	--------------------------------

Description

Adds a shaded interval to an existing density plot or regression plot.

Usage

```
shade(object, lim, label = NULL, col = col.alpha("black",
  0.15), border = NA, ...)
```

Arguments

<code>object</code>	A density or formula object that defines the density OR a matrix object that defines the plot region. See details.
<code>lim</code>	For a density, a vector of two values, indicating upper and lower bounds of interval, on the horizontal axis. For a plot region, a list of x-axis values corresponding to y-axis points in the matrix object.
<code>label</code>	Optional label to center in interval.
<code>col</code>	Color to shade the interval. Default is transparent gray.
<code>border</code>	Border of shaded region. Default is no border, NA.
<code>...</code>	Other parameters to pass to <code>polygon</code> , which actually draws the shaded region.

Details

This function uses [polygon](#) to draw a shaded region under a density curve or on a regression plot. The function assumes the plot already exists. See the examples below.

There are two plotting interfaces, for densities. First, if the density is plotted from kernel estimation, using perhaps [density](#), then the same density estimate should be passed as the first parameter. See example. Second, if the density is plotted from a series of x and y values, from perhaps a grid approximation, then a formula can be passed to define the curve. See example.

For plotting confidence regions on a regression plot, the matrix object should contain y-axis values defining the border of the region. The first row of the matrix should be the bottom of the region, and the second row should be the top. Each column should correspond to the x-axis values in `lim`. See example.

Author(s)

Richard McElreath

See Also

[density](#), [dens](#)

Examples

```
models <- seq( from=0 , to=1 , length.out=100 )
prior <- rep( 1 , 100 )
likelihood <- dbinom( 6 , size=9 , prob=models )
posterior <- likelihood * prior
posterior <- posterior / sum(posterior)

# using formula interface
plot( posterior ~ models , type="l" )
shade( posterior ~ models , c(0,0.5) )

# using density interface
samples <- sample( models , size=1e4 , replace=TRUE , prob=posterior )
plot( density(samples) )
shade( density(samples) , PCI(samples) )

# plotting a shaded confidence interval on a regression plot
data(cars)
m <- lm( dist ~ speed , cars )
p <- extract.samples( m )
x.seq <- seq( from=min(cars$speed)-1 , to=max(cars$speed)+1 , length.out=30 )
mu.ci <- sapply( x.seq , function(x) PI( p[,1] + p[,2]*x ) )
plot( dist ~ speed , cars )
abline( m )
shade( mu.ci , x.seq )
```

sim	<i>Simulate posterior observations</i>
-----	--

Description

Simulates posterior observations for map and map2stan model fits.

Usage

```
sim( fit , data , n=1000 , post , ll , refresh=0.1 , replace , ... )
```

Arguments

fit	Object of class map or map2stan
data	Optional list of data to compute predictions over
n	Number of samples to use
post	Samples from posterior. If missing, sim samples using n
...	Other parameters to pass to someone

Details

This function uses the model definition from a map or map2stan fit to simulate outcomes that average over the posterior distribution. It uses [link](#) internally to process any linear models. Then the correspond random number function is used, as defined by the model's likelihood.

The rethinking package defines a generic function sim, so methods can be defined for other model fit classes. I might eventually build methods for lm and glm.

Author(s)

Richard McElreath

See Also

[link](#), [map](#), [map2stan](#)

sim_train_test	<i>Simulate in-sample and out-of-sample model performance</i>
----------------	---

Description

Simulates in-sample and out-of-sample model performance for simple quap models, returning lppd in and out of sample, WAIC, LOOIC, and LOOCV.

Usage

```
sim_train_test(N = 20, k = 3, rho = c(0.15, -0.4), b_sigma = 100,
  WAIC = FALSE, LOOCV=FALSE , LOOIC=FALSE , cv.cores=1 ,
  return_model=FALSE )
```

```
# old function from 1st edition
sim.train.test( N=20 , k=3 , rho=c(0.15,-0.4) , b_sigma=100 ,
  DIC=FALSE , WAIC=FALSE, devbar=FALSE , devbarout=FALSE )
```

Arguments

N	Number of cases in simulated data
k	Number of parameters in model to fit to data
rho	Vector of correlations between predictors and outcome, in simulated data
b_sigma	Standard deviation of beta-coefficient priors
DIC	If TRUE, returns DIC
WAIC	If TRUE, returns WAIC
LOOIC	If TRUE, returns LOOIC as produced by LOO
LOOCV	If TRUE, returns LOOCV as produced by cv_quap
devbar	If TRUE, returns the average deviance in-sample
devbarout	If TRUE, returns average deviance out-of-sample
cv.cores	Number or cores to use for cross-validation
return_model	If TRUE, includes fit model in result

Details

This function simulates Gaussian data and then fits linear regression models to it, returning the lppd of the fit as produced by lppd (training, in-sample) and the deviance on a new sample, computed using the posterior from the training sample.

Author(s)

Richard McElreath

See Also

[lppd](#), [WAIC](#), [LOO](#), [cv_quap](#)

simplehist

*Simple histograms***Description**

Simple integer-valued histograms, for displaying count distributions.

Usage

```
simplehist( x , round=TRUE , ylab="Frequency" , ... )
```

Arguments

x	Vector of values to construct histogram from
round	When TRUE, rounds values in x before plotting
ylab	Label on vertical axis
...	Other parameters to pass to plot

Details

This function constructs clean histograms for count data. Non-integer data can be rounded to nearest integer before plotting. Internally, this function is little more than `plot(table(x))`.

Author(s)

Richard McElreath

See Also

[hist](#)

trankplot

*Diagnostic trace and rank histogram plots for MCMC output***Description**

The functions `trankplot` and `traceplot` display MCMC chain diagnostic plots. `trankplot` displays ranked histograms and `traceplot` shows the more traditional trace of the samples.

Usage

```
trankplot( object , bins=30 , pars , chains , col=rethink_palette , alpha=1 ,
  bg=col.alpha("black",0.15) , ask=TRUE , window , n_cols=3 , max_rows=5 ,
  lwd=1.5 , lp=FALSE , axes=FALSE , ... )
traceplot( object , pars , chains , col=rethink_palette , alpha=1 ,
  bg=col.alpha("black",0.15) , ask=TRUE , window , trim=100 , n_cols=3 ,
  max_rows=5 , lwd=0.5 , lp=FALSE , ... )
```

Arguments

object	A stanfit, ulam or map2stan object
bins	For trankplot, the number of histogram bins to use
pars	Optional character vector of parameters to display
chains	Optional integer vector of chains to display
col	Vector of colors to use for chains
alpha	Transparency
bg	Background color for warmup samples
ask	Interactive paging when TRUE
window	Optional range of samples to show
n_cols	Number of columns in display
max_rows	Maximum number of rows on each page
lwd	Line width
lp	Whether to include log_prob in display
axes	Whether to show axes on plots
trim	For traceplot, number of samples to trim for start. Helps with display, since early warmup samples typically very far from typical samples.
...	Additional arguments to pass to plot

Details

trankplot produces rank histograms of each chain, as described in Vehtari et al 2019 (see reference below). For each parameter, the samples from all chains are first ranked, using `rank_mat`. This returns a matrix of ranks, with the chains preserved. Then a histogram is built for each chain, using the same break points. These histograms are then overlain in the plot.

For healthy well-mixing chains, the histograms should be uniform. When there are spikes for some chains, especially in the low or high ranks, this suggests problems in exploring the posterior.

traceplot shows the sequential samples for each parameter and chain. This is the same information as the trankplot, but often much harder to see, given the volume of samples.

Author(s)

Richard McElreath

References

Vehtari, Gelman, Simpson, Carpenter, Bürkner. 2019. Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. <https://arxiv.org/abs/1903.08008>

Examples

```
## Not run:
library(rethinking)
data(chimpanzees)

d <- list(
  pulled_left = chimpanzees$pulled_left ,
  prosoc_left = chimpanzees$prosoc_left ,
  condition = as.integer( 2 - chimpanzees$condition ) ,
  actor = as.integer( chimpanzees$actor ) ,
  blockid = as.integer( chimpanzees$block )
)

m <- ulam(
  alist(
    # likelihood
    pulled_left ~ bernoulli(theta),

    # linear models
    logit(theta) <- A + BP*prosoc_left,
    A <- a + v[actor,1],
    BP <- bp + v[actor,condition+1],

    # adaptive prior
    vector[3]: v[actor] ~ multi_normal( 0 , Rho_actor , sigma_actor ),

    # fixed priors
    c(a,bp) ~ normal(0,1),
    sigma_actor ~ exponential(1),
    Rho_actor ~ lkjcorr(4)
  ) , data=d , chains=3 , cores=1 , sample=TRUE )

trankplot(m)

## End(Not run)
```

Trolley

Experimental data on ethical dilemmas

Description

These data comprise outcomes of experimental ethical dilemmas that are often called 'trolley' problems. Data kindly provided by Fiery Cushman.

Usage

```
data(Trolley)
```

Format

1. case: a code that combines treatment and story labels
2. response: participant's rating of appropriateness of action in story
3. order: integer order story appeared, within participant
4. id: participant id (factor)
5. age: participant's age in years
6. male: participant's gender; 1 for male, 0 for female
7. edu: participant's highest educational level
8. action: treatment code for story with action (1) or inaction (0)
9. intention: treatment code for intent (1) or lack of intent (0)
10. contact: treatment code for contact action (1) or lack of contact (0)
11. story: factor label for basic scenario modified by treatments
12. action2: alternative coding of action that is union of action and contact variables

References

Cushman et al. 2006. Psychological Science 17:1082–1089.

UFClefties

Ultimate Fighting Championship matches

Description

Outcomes of televised Ultimate Fighting Championship (UFC) matches, as a function of handedness of fighters. Data coded by Pollet et al (2013).

Usage

```
data(UFClefties)
```

Format

1. fight : Unique identifier for match
2. episode : Identifier for UFC episode
3. fight.in.episode : Order of fight in episode
4. fighter1.win : 1 if fighter 1 won the match; 0 if fighter 2 won
5. fighter1 : Unique identifier for fighter 1
6. fighter2 : Unique identifier for fighter 2
7. fighter1.lefty : 1 if fighter 1 was left handed; 0 otherwise
8. fighter2.lefty : 1 if fighter 2 was left handed; 0 otherwise

References

Pollet et al. 2013. Animal Behaviour 86:839–843.

ulam

*Build RStan models from formulas***Description**

Compiles lists of formulas into Stan model code. Allows for arbitrary fixed effect and mixed effect regressions. Allows for explicit typing of variables within the formula list. Much more flexible than `map2stan`.

Usage

```
ulam( flist , data , pars , pars_omit , start , chains=1 , cores=1 , iter=1000 ,
      control=list(adapt_delta=0.95) , distribution_library=ulam_dists ,
      macro_library=ulam_macros , custom , declare_all_data=TRUE , log_lik=FALSE ,
      sample=TRUE , messages=TRUE , pre_scan_data=TRUE , coerce_int=TRUE ,
      sample_prior=FALSE , file=NULL , cmdstan=FALSE , threads=1 ,
      stanc_options=list("O1") , ... )
```

Arguments

<code>flist</code>	A formula or list of formulas that define the likelihood and priors. Can also pass in a quap or a previous <code>ulam</code> model fit. See details.
<code>data</code>	A data frame or list containing the data
<code>pars</code>	Optional: character vector of parameters to return samples for
<code>pars_omit</code>	Optional: character vector of parameters to exclude from samples
<code>start</code>	Optional. Either: (1) a named list specifying parameters and their initial values or (2) a function to return such a named list.
<code>chains</code>	Number of independent chains to sample from
<code>cores</code>	Number of processor cores to distribute chains over.
<code>iter</code>	Number of iterations of sampling. By default, half of these iterations are warmup.
<code>control</code>	Optional list of control parameters for <code>stan</code> . Default increases target acceptance rate (<code>adapt_delta</code>) to 0.95.
<code>distribution_library</code>	List of distribution templates.
<code>macro_library</code>	List of function and distribution macros.
<code>custom</code>	Optional list of custom resources. See details and examples.
<code>declare_all_data</code>	When <code>TRUE</code> , all variables in the data list are declared in the Stan model code. When <code>FALSE</code> , only used variables are declared.
<code>log_lik</code>	Return log likelihood of each observation in samples. Used for calculating WAIC and LOO.
<code>sample</code>	If <code>FALSE</code> , builds Stan code without sampling

<code>messages</code>	Show various warnings and informational messages
<code>pre_scan_data</code>	Scan data at start to (1) check for variables that are integer but not type integer and (2) strip any <code>scale()</code> attributes
<code>coerce_int</code>	If <code>pre_scan_data</code> , forces integer variables to be type integer
<code>sample_prior</code>	If TRUE, removes data probabilities from model to sample only prior distribution of parameters. Used by corresponding <code>extract.prior</code> method.
<code>messages</code>	If TRUE, prints various internal steps to help with debugging
<code>file</code>	If character string X, loads X.rds instead of fitting when exists; otherwise saves result to X.rds
<code>cmdstan</code>	When TRUE, uses <code>cmdstanr</code> instead of <code>rstan</code> to run chains. To make <code>cmdstan</code> the default engine, use <code>set_ulam_cmdstan(TRUE)</code> .
<code>threads</code>	When <code>threads > 1</code> , attempts to multithread individual chains using Stan's <code>reduce_sum</code> function. Requires <code>cmdstan=TRUE</code> .
<code>...</code>	Additional arguments to pass to stan

Details

`ulam` provides a slim version of Stan syntax that omits blocks but still allows for explicit variable types and dimensions. The basic model formula is the same as `map2stan`, but the syntax does not assume a GLMM structure. This allows it to be more flexible, but it also means more mistakes are possible. With great power comes great responsibility.

The function of a model formula is to related the variables to one another. There are three types of variables: (1) data, (2) parameters, and (3) local variables. Model formulas are composed of multiple lines. Each line defines a variable using either a distributional assumption like:

```
y ~ normal( mu , sigma )
```

or rather a determinsitic assignment like:

```
mu <- a + b*x
```

The basic structure of such a definition is:

```
type[dimension]: name[dimension] ~ distribution( arguments )
```

The type delcaration is optional. So the most basic defintion can be just `y ~ bernoulli(theta)`, but a full declaration can be more detailed, when necessary. The examples below show how matrix variables can be defined in this syntax.

For determinstic assignments like:

```
mu <- a + b*x
```

It is also possible to use a control word to specify how the values are returned. Using `save>` returns the values in the posterior samples. For example:

```
save> mu <- a + b*x
```

will return `mu` for each case and posterior sample. This works by duplicating the code in both the model block, where it is used to compute the log-probability, and in generated quantities.

It is also possible to use `gq` to evaluate the assignment only after sampling, in Stan's generated quantities block. This is useful for derived values that are not needed in computing the posterior but

may be useful afterwards. For example, contrasts could be calculated this way. In the examples, the line:

```
gq> bp_diff <- bp[1] - bp[2]
```

is used to calculate the posterior distribution of the difference between the two parameters. The code is added to Stan's generated quantities, so that it doesn't slow down the model block.

The control tag `transpars` can be used to place an assignment in Stan's transformed parameters block. Keep in mind that any other intermediate calculations must also be placed in the same block. Finally, `transdata` places the assignment in the transformed data block. This means it will only execute once, before sampling begins. But it also means that the values will not be available post-sampling for helper functions like `link`. As such, it will usually be better to transform data before passing it into `ulam`.

When `cmdstan=TRUE`, the `cmdstanr` package will be used instead `rstan` to compile and sample from models. This is generally superior, as more recent versions of Stan can be used this way. But some features are not yet implemented, such as passing custom inits to the chains. To make `cmdstan` the default engine, use `set_ulam_cmdstan(TRUE)`. You can then ignore the `cmdstan` argument when calling `ulam`.

The use of `cmdstan=TRUE` is also the only way to currently use multi-threading of individual chains, using the `threads` argument. When `cmdstan=TRUE` and `threads` is set greater than 1, `ulam` will try to recode the model so that each chain is spread over multiple cores. This can easily halve sampling time. At the moment, this only works for models with a single outcome variable.

Methods are defined for [extract.samples](#), [extract.prior](#), [link](#), [sim](#), [compare](#), `coef`, `summary`, `logLik`, `lppd`, `vcov`, `nobs`, `deviance`, `WAIC`, `PSIS`, `plot`, `traceplot`, `trankplot`, `pairs`, and `show`.

Value

Returns an object of class `ulam` with the following slots.

<code>call</code>	The function call
<code>model</code>	Stan model code
<code>stanfit</code>	<code>stanfit</code> object returned by stan
<code>coef</code>	The posterior means
<code>vcov</code>	k-by-1 matrix containing the variance of each of k variables in posterior
<code>data</code>	The data
<code>start</code>	List of starting values that were used in sampling
<code>pars</code>	Parameter names monitored in samples
<code>formula</code>	Formula list from call
<code>formula_parsed</code>	List of parsed formula information. Useful mainly for debugging. Needed by helper functions.

Author(s)

Richard McElreath

See Also

[quap](#), [map2stan](#), [stan](#)

Examples

```
## Not run:
library(rethinking)
data(chimpanzees)

# don't want any variables with NAs
# also recode condition to an index {1,0} -> {1,2}
d <- list(
  pulled_left = chimpanzees$pulled_left ,
  prosoc_left = chimpanzees$prosoc_left ,
  condition = as.integer( 2 - chimpanzees$condition ) ,
  actor = as.integer( chimpanzees$actor ) ,
  blockid = as.integer( chimpanzees$block )
)

# simple logistic regression
m1 <- ulam(
  alist(
    pulled_left ~ bernoulli(theta),
    logit(theta) <- a + bp[condition]*prosoc_left ,
    a ~ normal(0,4),
    bp[condition] ~ normal(0,1)
  ) ,
  data=d, chains=2, cores=1 , sample=TRUE )

precis(m1,depth=2)
plot(m1,depth=2)
pairs(m1)

# same model, but save theta so it is return in samples
# note 'save>' in second line of formula
m1b <- ulam(
  alist(
    pulled_left ~ bernoulli(theta),
    save> logit(theta) <- a + bp[condition]*prosoc_left ,
    a ~ normal(0,4),
    bp[condition] ~ normal(0,1)
  ) ,
  data=d, chains=2, cores=1 , sample=TRUE )

# same model, but use gq to compute contrast between conditions
# note that order does matter. bp_diff should come before bp[] is defined
m1c <- ulam(
  alist(
    pulled_left ~ bernoulli(theta),
    logit(theta) <- a + bp[condition]*prosoc_left ,
    gq> bp_diff <- bp[1] - bp[2],
    a ~ normal(0,4),
    bp[condition] ~ normal(0,1)
  ) ,
  data=d, chains=2, cores=1 , sample=TRUE )
```

```

# can also transform data inside model, using transdata> tag.
# this is more efficient, because it only evaluates once, not during sampling.
# for example, this constructs prosoc_right variable:
m1d <- ulam(
  alist(
    pulled_left ~ bernoulli(theta),
    logit(theta) <- a + bp[condition]*prosoc_right ,
    transdata> prosoc_right <- 1 - prosoc_left,
    a ~ normal(0,4),
    bp[condition] ~ normal(0,1)
  ) ,
  data=d, chains=2, cores=1 , sample=TRUE )

# now model with varying intercepts on actor
m2 <- ulam(
  alist(
    pulled_left ~ bernoulli(theta),
    logit(theta) <- a + aj[actor] + bp[condition]*prosoc_left,
    aj[actor] ~ normal( 0 , sigma_actor ),
    a ~ normal(0,4),
    bp[condition] ~ normal(0,1),
    sigma_actor ~ exponential(1)
  ) ,
  data=d, chains=2 , cores=1 , sample=TRUE )

precis(m2)
plot(m2)

# varying intercepts on actor and experimental block
m3 <- ulam(
  alist(
    pulled_left ~ bernoulli(theta),
    logit(theta) <- a + aj[actor] + ak[blockid] + bp[condition]*prosoc_left,
    aj[actor] ~ normal( 0 , sigma_actor ),
    ak[blockid] ~ normal( 0 , sigma_block ),
    a ~ dnorm(0,4),
    bp[condition] ~ dnorm(0,1),
    sigma_actor ~ exponential(1),
    sigma_block ~ exponential(1)
  ) ,
  data=d, chains=2 , cores=1 , sample=TRUE )

precis(m3)
summary(m3)
plot(m3)

#####
# varying slopes models

# varying slopes on actor
# also demonstrates use of multiple linear models
# see Chapter 13 for discussion
m3 <- ulam(

```

```

alist(
  # likelihood
  pulled_left ~ bernoulli(theta),

  # linear models
  logit(theta) <- A + BP*prosoc_left,
  A <- a + v[actor,1],
  BP <- bp + v[actor,condition+1],

  # adaptive prior
  vector[3]: v[actor] ~ multi_normal( 0 , Rho_actor , sigma_actor ),

  # fixed priors
  c(a,bp) ~ normal(0,1),
  sigma_actor ~ exponential(1),
  Rho_actor ~ lkjcorr(4)
) , data=d , chains=3 , cores=1 , sample=TRUE )

# same model but with non-centered parameterization
# see Chapter 13 for explanation and more elaborate example
m4 <- ulam(
  alist(
    # likelihood
    pulled_left ~ bernoulli(theta),

    # linear models
    logit(theta) <- A + BP*prosoc_left,
    A <- a + v[actor,1],
    BP <- bp + v[actor,condition+1],

    # adaptive prior
    matrix[actor,3]: v <- compose_noncentered( sigma_actor , L_Rho_actor , z ),
    matrix[3,actor]: z ~ normal( 0 , 1 ),

    # fixed priors
    c(a,bp) ~ normal(0,1),
    vector[3]: sigma_actor ~ exponential(1),
    cholesky_factor_corr[3]: L_Rho_actor ~ lkj_corr_cholesky( 4 )
  ) , data=d , chains=3 , cores=1 , sample=TRUE )

# same as m5, but without hiding the construction of v
m5 <- ulam(
  alist(
    # likelihood
    pulled_left ~ bernoulli(theta),

    # linear models
    logit(theta) <- A + BP*prosoc_left,
    A <- a + v[actor,1],
    BP <- bp + v[actor,condition+1],

    # adaptive prior
    matrix[actor,3]: v <- t(diag_pre_multiply( sigma_actor , L_Rho_actor ) * z),

```

```

matrix[3,actor]: z ~ normal( 0 , 1 ),

# fixed priors
c(a,bp,bpc) ~ normal(0,1),
vector[3]: sigma_actor ~ exponential(1),
cholesky_factor_corr[3]: L_Rho_actor ~ lkj_corr_cholesky( 4 )
) , data=d , chains=3 , cores=1 , sample=TRUE )

## End(Not run)

```

WaffleDivorce

Waffle House and marriage statistics

Description

Data for the individual States of the United States, describing number of Waffle House diners and various marriage and demographic facts.

Usage

```
data(WaffleDivorce)
```

Format

1. Location : State name
2. Loc : State abbreviation
3. Population : 2010 population in millions
4. MedianAgeMarriage: 2005-2010 median age at marriage
5. Marriage : 2009 marriage rate per 1000 adults
6. Marriage.SE : Standard error of rate
7. Divorce : 2009 divorce rate per 1000 adults
8. Divorce.SE : Standard error of rate
9. WaffleHouses : Number of diners
10. South : 1 indicates Southern State
11. Slaves1860 : Number of slaves in 1860 census
12. Population1860 : Population from 1860 census
13. PropSlaves1860 : Proportion of total population that were slaves in 1860

References

1860 census data from <http://mapserver.lib.virginia.edu>. Marriage and divorce rates from 2009 American Community Survey (ACS). Waffle House density data from wafflehouse.com (retrieved January 2012).

WAIC	<i>Information Criteria and Pareto-Smoothed Importance Sampling Cross-Validation</i>
------	--

Description

Computes WAIC, DIC, and PSIS cross validation for quap, map2stan, ulam model fits. In addition, WAIC and PSIS can be calculated for stan model fits (see details).

Usage

```
WAIC( object , n=1000 , refresh=0.1 , pointwise=FALSE , ... )
PSIS( object , n=1000 , refresh=0.1 , pointwise=FALSE , ... )
DIC( object , ... )
```

Arguments

object	Object of class map or map2stan
n	Number of samples to use in computing WAIC. Set to n=0 to use all samples in map2stan fit
refresh	Refresh interval for progress display. Set to refresh=0 to suppress display.
pointwise	If TRUE, return a vector of WAIC values for each observation. Useful for computing standard errors.
...	Other parameters to pass to specific methods

Details

These functions use the samples and model definition to compute the Widely Applicable Information Criterion (WAIC), Deviance Information Criterion (DIC), or Pareto-smoothed importance-sampling cross-validation estimate (PSIS).

WAIC is an estimate of out-of-sample relative K-L divergence (KLD), and it is defined as:

$$WAIC = -2(lppd - pWAIC)$$

Components lppd (log pointwise predictive density) and pWAIC (the effective number of parameters) are reported as attributes. See Gelman et al 2013 for definitions and formulas. This function uses the variance definition for pWAIC.

PSIS is another estimate of out-of-sample relative K-L divergence. It is computed by the loo package. See Vehtari et al 2015 for definitions and computation.

In practice, WAIC and PSIS are extremely similar estimates of KLD.

Both WAIC and PSIS have methods for stanfit models, provided the posterior contains a log-likelihood matrix (samples on rows, observations on columns) named log_lik. See example.

Author(s)

Richard McElreath

References

- Watanabe, S. 2010. Asymptotic equivalence of Bayes cross validation and Widely Applicable Information Criterion in singular learning theory. *Journal of Machine Learning Research* 11:3571-3594.
- Gelman, A., J. Hwang, and A. Vehtari. 2013. Understanding predictive information criteria for Bayesian models.
- Vehtari, A., A. Gelman, and J. Gabry. 2015. Efficient implementation of leave-one-out cross-validation and WAIC for evaluating fitted Bayesian models.

See Also

[quap](#), [ulam](#), [link](#), [loo](#)

Examples

```
## Not run:
library(rethinking)
data(chimpanzees)
d <- chimpanzees
dat <- list(
  y = d$pulled_left,
  prosoc = d$prosoc_left,
  condition = d$condition,
  N = nrow(d)
)

m1s_code <- '
data{
  int<lower=1> N;
  int y[N];
  int prosoc[N];
}
parameters{
  real a;
  real bP;
}
model{
  vector[N] p;
  bP ~ normal( 0 , 1 );
  a ~ normal( 0 , 10 );
  for ( i in 1:N ) {
    p[i] = a + bP * prosoc[i];
  }
  y ~ binomial_logit( 1 , p );
}
generated quantities{
  vector[N] p;
  vector[N] log_lik;
  for ( i in 1:N ) {
    p[i] = a + bP * prosoc[i];
    log_lik[i] = binomial_logit_lpmf( y[i] | 1 , p[i] );
  }
}
```

```
}  
,  
  
m1s <- stan( model_code=m1s_code , data=dat , chains=2 , iter=2000 )  
  
WAIC(m1s)  
  
PSIS(m1s)  
  
## End(Not run)
```

Wines2012

Wine tasting scores, Princeton NJ 2012

Description

Numerical scores from 9 judges, both French and American, testing 20 different wines, both French and American. Judges were blind to the identity of each wine during the testing.

Usage

```
data(Wines2012)
```

Format

1. judge : Name of judge
2. flight : white or red group of 10 wines
3. wine : Wine ID
4. score : Numerical score of wine
5. wine.amer : 1 for American wines from New Jersey
6. judge.amer : 1 for American judges

References

Raw data from <http://www.liquidasset.com/report161.html>

Index

- * **classes**
 - map2stan-class, [53](#)
- * **datasets**
 - reedfrogs, [67](#)
- * **package**
 - rethinking-package, [3](#)
- * **rethinking**
 - rethinking-package, [3](#)
- Achehunting, [4](#)
- AMTL, [4](#)
- AMTL_short (AMTL), [4](#)
- axis_unscale, [5](#)
- bangladesh, [6](#)
- Boxes, [7](#)
- chainmode, [8](#)
- check_index (coerce_index), [13](#)
- cherry_blossoms, [8](#)
- chimpanzees, [10](#)
- coeftab, [11](#), [12](#)
- coeftab_plot, [12](#)
- coerce_index, [13](#)
- col.alpha, [14](#)
- col.desat (col.alpha), [14](#)
- col.dist (col.alpha), [14](#)
- col2rgb, [14](#)
- compare, [15](#), [31](#), [48](#), [81](#)
- contour, [16](#)
- contour_xyz, [16](#)
- coordinates, [26](#)
- Crofoot, [16](#)
- cv_quap, [17](#), [74](#)
- dagitty::paths, [26](#)
- dbetabinom, [18](#)
- dens, [19](#), [72](#)
- density, [20](#), [56](#), [72](#)
- deviance, map2stan-method (map2stan-class), [53](#)
- dgamma, [28](#)
- dgam pois, [20](#)
- DIC, [65](#)
- DIC (WAIC), [86](#)
- DIC, map2stan-method (map2stan-class), [53](#)
- Dinosaurs, [21](#)
- Dissertations, [22](#)
- dlk jcorr, [23](#)
- dmvnorm2, [24](#)
- dordlogit, [25](#)
- dotchart, [12](#)
- drawdag, [25](#)
- drawopenpaths (drawdag), [25](#)
- dstudent, [27](#)
- dzagamma2, [28](#)
- dzibinom, [29](#)
- dzipois, [29](#)
- ensemble, [30](#), [43](#), [44](#), [48](#)
- extract.prior, [81](#)
- extract.prior (extract.samples), [31](#)
- extract.samples, [31](#), [43](#), [48](#), [81](#)
- extract.samples, map2stan-method (map2stan-class), [53](#)
- Fish, [33](#)
- foxes, [33](#)
- glimmer, [34](#), [48](#)
- glm, [34](#)
- graphLayout, [26](#)
- grau (col.alpha), [14](#)
- hist, [75](#)
- HMC2, [35](#)
- HMC_2D_sample (HMC2), [35](#)
- Hoogland, [37](#)
- Howell, [38](#)
- Howell11 (Howell), [38](#)
- Howell12 (Howell), [38](#)

- HPDI, [38](#), [59](#)
- HPDinterval, [20](#), [39](#)
- Hurricanes, [39](#)
- ICweights (compare), [15](#)
- image, [40](#)
- image_xyz, [40](#)
- islands (Kline), [40](#)
- islandsDistMatrix (Kline), [40](#)
- Kline, [40](#)
- Kline2 (Kline), [40](#)
- KosterLeckie, [41](#)
- Laffer, [42](#)
- link, [31](#), [48](#), [58](#), [73](#), [81](#), [87](#)
- link (link-methods), [43](#)
- link, lm-method (link-methods), [43](#)
- link, map-method (link-methods), [43](#)
- link, map2stan-method (link-methods), [43](#)
- link-methods, [43](#)
- log_sum_exp, [45](#)
- logLik, map2stan-method (map2stan-class), [53](#)
- LOO, [74](#)
- LOO (WAIC), [86](#)
- loo, [87](#)
- lppd, [74](#)
- map, [44](#), [48](#), [56](#), [73](#)
- map (quap), [65](#)
- map2stan, [35](#), [44](#), [45](#), [53](#), [56](#), [57](#), [69](#), [73](#), [81](#)
- map2stan-class, [53](#)
- mclapply, [54](#), [68](#), [69](#)
- mcreplicate, [53](#)
- milk, [54](#)
- Moralizing_gods, [55](#)
- mvnorm, [32](#), [71](#)
- nobs, map2stan-method (map2stan-class), [53](#)
- optim, [66](#)
- pairs, map2stan-method (map2stan-class), [53](#)
- pairs.map (pairs.map2stan), [56](#)
- pairs.map2stan, [56](#)
- Panda_nuts, [56](#)
- parallel, [46](#)
- parLapply, [68](#)
- PCI (HPDI), [38](#)
- PI (HPDI), [38](#)
- plot, [76](#)
- plot, map2stan-method (map2stan-class), [53](#)
- plot.map2stan, [57](#)
- polygon, [72](#)
- pordlogit (dordlogit), [25](#)
- postcheck, [43](#), [44](#), [58](#)
- precis, [58](#)
- Primates301, [59](#)
- progbar, [63](#)
- PrussianHorses, [64](#)
- PSIS (WAIC), [86](#)
- quantile, [39](#)
- quap, [3](#), [4](#), [17](#), [65](#), [81](#), [87](#)
- rbeta, [18](#)
- rbetabinom (dbetabinom), [18](#)
- rbinom, [18](#)
- reedfrogs, [67](#)
- replicate, [53](#), [54](#)
- resample, [48](#), [68](#)
- rethinking (rethinking-package), [3](#)
- rethinking-package, [3](#)
- rgam pois (dgam pois), [20](#)
- rgb, [14](#)
- rgb2hsv, [14](#)
- Rinder, [69](#)
- rlkjcorr (dlkjcorr), [23](#)
- rmvnorm2 (dmvnorm2), [24](#)
- rordlogit (dordlogit), [25](#)
- rstudent (dstudent), [27](#)
- rzibinom (dzibinom), [29](#)
- rzipois (dzipois), [29](#)
- sample.naive.posterior (sample.qa.posterior), [70](#)
- sample.qa.posterior, [70](#)
- sflist2stanfit, [68](#), [69](#)
- shade, [71](#)
- show, map2stan-method (map2stan-class), [53](#)
- sim, [31](#), [43](#), [44](#), [48](#), [58](#), [73](#), [81](#)
- sim, lm-method (sim), [73](#)
- sim, map-method (sim), [73](#)
- sim, map2stan-method (map2stan-class), [53](#)

`sim, map2stan-method (sim)`, [73](#)
`sim-methods (sim)`, [73](#)
`sim.train.test (sim_train_test)`, [74](#)
`sim_train_test`, [74](#)
`simplehist`, [75](#)
`stan`, [46](#), [48](#), [53](#), [80](#), [81](#)
`stancode, map2stan-method`
 (`map2stan-class`), [53](#)
`summary, map2stan-method`
 (`map2stan-class`), [53](#)

`traceplot (trankplot)`, [75](#)
`trankplot`, [75](#)
`Trolley`, [77](#)

`UFClefties`, [78](#)
`ulam`, [3](#), [4](#), [79](#), [87](#)

`vcov, map2stan-method (map2stan-class)`,
 [53](#)

`WaffleDivorce`, [85](#)
`WAIC`, [43](#), [74](#), [86](#)
`WAIC, map2stan-method (map2stan-class)`,
 [53](#)
`Wines2012`, [88](#)