

Package: INLAcircular (via r-universe)

October 29, 2026

Type Package

Title Bayesian Joint Circular Regression Using 'INLA'

Version 1.0.0

Description Provides a Bayesian regression framework for joint models with circular responses and covariates using integrated nested Laplace approximations, following Ye, Van Niekerk and Rue (2026) <[doi:10.48550/arXiv.2602.08413](https://doi.org/10.48550/arXiv.2602.08413)>. The package supports link-adjusted von Mises likelihoods, shared latent predictors, structured latent effects, and optional LKJ-based multivariate covariance models. It also provides compiled circular densities and d/p/q/r penalized-complexity priors for von Mises, cardioid, and wrapped Cauchy concentration parameters as developed by Ye, Van Niekerk and Rue (2026) <[doi:10.1177/1471082X261453677](https://doi.org/10.1177/1471082X261453677)>.

License GPL (>= 3)

URL <https://github.com/XiangYEstats/INLAcircular>

BugReports <https://github.com/XiangYEstats/INLAcircular/issues>

Depends R (>= 4.1.0)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Imports dplyr, stats, utils

Suggests circular, graphpcor, INLA (>= 25.08.21), INLAtools, knitr, pkgload, rmarkdown

VignetteBuilder knitr, rmarkdown

Additional_repositories <https://inla.r-inla-download.org/R/stable>

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

Date/Publication 2026-10-18 11:55:34 UTC

RemoteUrl <https://github.com/xiangyestats/inlacircular>

RemoteRef HEAD
RemoteSha a6f1ed1e46b49150120cbb4452598e5ac383a03c

Contents

bessel_i	2
circular_densities	3
covariate	5
index	6
inla	7
inlacc	9
intercept	11
lavm	12
lavm.cloglike	14
lavm.link	15
likelihood	16
LKJcc	17
pc_card	18
pc_card0	20
pc_vm0	21
pc_vminf	22
pc_wc	24
print.inlacc_summary	25
print.inlacc_time	26
summary.inlacc	27
von_mises	28
wind_newyork	29
Index	31

bessel_i	<i>Modified Bessel Function of the First Kind</i>
----------	---

Description

Evaluates the modified Bessel function of the first kind using native C routines adapted from the GNU Scientific Library (GSL). R supplies only the interface; argument checking and numerical computation take place in C.

Usage

bessel_i(x, nu, expon.scaled = FALSE)

Arguments

<code>x</code>	Numeric vector. Negative values are supported for integer orders. Missing values are propagated.
<code>nu</code>	One finite, nonnegative numeric value indicating the order. Fractional orders are supported for nonnegative <code>x</code> .
<code>expon.scaled</code>	Logical scalar; if TRUE, return $I_\nu(x) \exp(- x)$. Defaults to FALSE.

Details

Negative `x` with a fractional order produces NaN and a warning. For an integer order n , $I_n(-x) = (-1)^n I_n(x)$.

At zero, the result is one for order zero and zero for positive orders. At positive infinity, the result is zero when scaled and infinity otherwise. Unscaled results can overflow for large arguments; use `expon.scaled = TRUE` when the scaled value is needed. For very large orders, cancellation can reduce the accuracy of an unscaled result. A warning is issued when the estimated loss is significant.

Value

A numeric vector of calculated Bessel values.

Examples

```
bessel_i(1, nu = 0.5)
bessel_i(c(0, 1, 5), nu = 0)
bessel_i(1000, nu = 0, expon.scaled = TRUE)
```

circular_densities	<i>Cardioid and Wrapped Cauchy Densities</i>
--------------------	--

Description

Compiled density functions for two circular distributions, parameterized by mean direction `mu` and concentration `kappa`.

Usage

```
dcardioid(x, mu, kappa, log = FALSE)
dwrappedcauchy(x, mu, kappa, log = FALSE)
dcard(x, mu, kappa, log = FALSE)
dwc(x, mu, kappa, log = FALSE)
```

Arguments

x	Vector of angles in radians.
mu	Vector of mean directions in radians.
kappa	Vector of concentration parameters. For the cardioid density, $0 \leq \kappa \leq 0.5$; for the wrapped Cauchy density, $0 \leq \kappa < 1$.
log	Logical scalar; if TRUE, return log-densities.

Details

The cardioid density is

$$f(x \mid \mu, \kappa) = \frac{1 + 2\kappa \cos(x - \mu)}{2\pi}, \quad 0 \leq \kappa \leq \frac{1}{2}.$$

The wrapped Cauchy density is

$$f(x \mid \mu, \kappa) = \frac{1 - \kappa^2}{2\pi\{1 + \kappa^2 - 2\kappa \cos(x - \mu)\}}, \quad 0 \leq \kappa < 1.$$

Angles are periodic, so adding an integer multiple of 2π to x or mu does not change the result. The calculations are performed in compiled C code. The wrapped Cauchy denominator is evaluated in a cancellation-resistant form near $\kappa = 1$.

At the limiting wrapped Cauchy boundary $\kappa = 1$, the compiled function returns infinite density at the mean direction and zero elsewhere. This describes the point-mass limit; finite-parameter likelihoods should use $\kappa < 1$.

Value

A numeric vector of densities or log-densities. The arguments are recycled to a common length.

Examples

```

angles <- seq(-pi, pi, length.out = 5)
dcardioid(angles, mu = 0, kappa = 0.25)
dwrappedcauchy(angles, mu = 0, kappa = 0.7)

# Short aliases give the same results.
identical(
  dcard(angles, mu = 0, kappa = 0.25),
  dcardioid(angles, mu = 0, kappa = 0.25)
)
```

covariate

*Define a covariate or copied predictor in an `inlacc()` formula***Description**

`covariate()` is a formula marker with two modes. For an ordinary data variable it defines a fixed effect. When `var` is also a response in another block, it can define an INLA copy-scaling coefficient for that response's latent predictor.

Usage

```
covariate(
  var,
  name = NULL,
  mean = NULL,
  sd = NULL,
  initial = NULL,
  fixed = FALSE,
  predictor = NULL
)
```

Arguments

<code>var</code>	An unquoted data-column name or numeric expression. This is an argument name, not a separate <code>var()</code> function.
<code>name</code>	Optional coefficient and display name.
<code>mean</code>	Optional Gaussian prior mean.
<code>sd</code>	Optional Gaussian prior standard deviation.
<code>initial</code>	Optional initial value for a copied predictor's INLA scaling coefficient. It is ignored for an ordinary fixed covariate. The copied coefficient default is 0.
<code>fixed</code>	Logical; for a copied predictor, whether to fix the scaling coefficient at <code>initial</code> . It is ignored for an ordinary fixed covariate.
<code>predictor</code>	NULL or one logical value. The default NULL automatically uses an INLA copy term when <code>var</code> is an untransformed response from another likelihood block. TRUE explicitly requests that behavior and errors unless <code>var</code> is such a response. FALSE always uses the observed value as an ordinary fixed covariate.

Details

For an ordinary fixed effect, `mean` and `sd` independently override `control.fixed`; omitted values inherit the global settings. For a copied predictor, both must be supplied to customize the scaling prior. If either is missing, the current implementation uses its default $N(0, 31.62^2)$ prior. Set `control.predictor = list(compute = TRUE)` in `inlacc()` when fitted latent predictors are required. A response used as a bare formula term is also copied automatically; write `covariate(response, predictor = FALSE)` to use its observed values instead. Transformed response expressions are ordinary observed fixed effects unless `predictor = TRUE`, which is invalid because a transformed latent predictor cannot be copied by this interface.

Value

NULL, invisibly; `inlacc()` interprets the unevaluated call.

See Also

`likelihood()`, `intercept()`, `inlacc()`

Examples

```
likelihood(
  y ~ covariate(z, name = "beta_z", mean = 0, sd = 2),
  family = "gaussian"
)

# In a joint model, `angle` is copied automatically because it is another
# likelihood block's response. Use predictor = TRUE to make that explicit.
likelihood(
  y ~ covariate(
    angle,
    name = "beta_angle_predictor",
    mean = 0,
    sd = 1,
    initial = 0,
    fixed = FALSE,
    predictor = TRUE
  ),
  family = "gaussian"
)
```

index

Map observations to a latent process in `inlacc()`

Description

Create an observation-to-process mapping for an INLA `f()` term. Pass one mapping, or a list of mappings, through the `latent.index` argument of `inlacc()`.

Usage

```
index(var, data.id, process.id = NULL)
```

Arguments

<code>var</code>	Character string naming the first argument of the corresponding bare <code>f()</code> term, for example <code>"hour_effect"</code> . Quote the name.
<code>data.id</code>	Observation-level node identifiers. This should normally have <code>nrow(data)</code> entries. The special value <code>"likelihood"</code> assigns the likelihood-block number for an ordinary shared effect.

`process.id` Optional complete ordered node set. When supplied for a matching first `f()` index, `inlacc()` injects it as the INLA values argument, replacing an existing values argument and allowing unobserved process nodes. If `NULL`, no explicit values vector is injected and INLA infers the represented values from `data.id`.

Details

If `var` is not used as an `f()` first, group, or replicate argument, `inlacc()` ignores the mapping with a warning. Without a mapping or same-named data column, a discovered index is generated as `seq_len(nrow(data))`. A mapping may provide `data.id` for a first, group, or replicate variable, but `process.id` is injected only when `var` matches the first argument of `f()`.

For `model = "iidkd_LKJ"`, the special form `index(var, data.id = replicate_ids, process.id = "likelihood")` has a different meaning: `data.id` supplies positive integer replicate IDs and the component index is generated from likelihood order. See `LKJcc()`.

Value

A list containing `var`, `data.id`, and `process.id`.

See Also

`inlacc()`, `LKJcc()`

Examples

```
hour_index <- index(
  var = "hour_effect",
  data.id = c(1, 2, 2, 4),
  process.id = 1:4
)
```

inla

Run INLA with the INLAcircular LAvM likelihood

Description

`inla()` is a compatibility layer over `INLA::inla()`. Calls without `family = "lavm"` are forwarded unchanged. For LAvM calls, it installs the package's C `cloglike`, converts the response to an INLA matrix-valued response, and evaluates the `pc.vm0` or `pc.vminf` concentration prior in the same C module used by `inlacc()`.

Usage

```
inla(...)
```

Arguments

... Arguments accepted by `INLA::inla()`. For one family = "lavm", use `control.family = list(link = ..., hyper = list(kappa = ...))`; the nested form `list(list(...))` is also accepted. With multiple families, `control.family` must contain exactly one list per family.

Details

Loading `INLAcircular` after `INLA` makes this function available as `inla()`. The underlying likelihood sent to `INLA` is "cloglike", which is necessary because external packages cannot add a new compiled family or prior keyword to an already-built `INLA` executable. All non-LAvM arguments retain `INLA`'s own matching and evaluation rules.

Within `hyper$kappa`, `initial` is on the internal $\log(\kappa)$ scale. The defaults are `initial = 6`, `prior = "pc.vminf"`, `param = c(0.5, 0.5)`, and `fixed = FALSE`.

When multiple families are requested and at least one is "lavm", the formula response must evaluate to a matrix, data frame, or list with one component per family. LAvM components are converted with `INLA::inla.mdata()`; other families and controls are retained.

For any call containing LAvM, missing `control.inla$cmin` and `control.inla$compute.initial.values` fields are filled with 0 and TRUE, respectively. User-supplied fields and values take precedence.

The returned object gains an `INLAcircular` list recording the requested families, effective internal families, and LAvM prior labels. LAvM concentration summaries and marginals are normally transformed from internal $\log(\kappa)$ to natural-scale κ and renamed.

For a literal namespaced `INLA::inla()` call, construct the external likelihood explicitly with `lavm.cloglike()`, convert the response with `INLA::inla.mdata()`, and use `family = "cloglike"`. A namespaced call with `family = "lavm"` bypasses this compatibility layer.

Value

The fitted object returned by `INLA::inla()`, augmented as described in Details when LAvM is requested.

See Also

[lavm.cloglike\(\)](#), [inlacc\(\)](#)

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  n <- 30
  z <- rnorm(n)
  y <- rlavm(n, eta = 1 + 1.5 * z, kappa = 100)

  fit <- INLAcircular::inla(
    y ~ z,
    family = "lavm",
    data = data.frame(y = y, z = z),
    num.threads = "2:1",
```

```

    control.family = list(
      link = "inverse.tangent",
      hyper = list(
        kappa = list(
          initial = 6,
          prior = "pc.vminf",
          param = c(0.5, 0.5),
          fixed = FALSE
        )
      )
    )
  )
  fit$summary.fixed
}

```

inlacc

Fit single or joint circular models with INLA

Description

`inlacc()` assembles one or more `likelihood()` blocks into a stacked INLA model. It supports ordinary INLA likelihoods and latent effects, the package's compiled LAvM likelihood, copied response predictors, explicit latent-process index mappings, and optional graphpcor-backed LKJ covariance components.

Usage

```

inlacc(
  model,
  data,
  latent.index = NULL,
  LKJ.eta = 5,
  control.family = NULL,
  control.fixed = NULL,
  control.inla = list(cmin = 0, compute.initial.values = TRUE),
  metrics = FALSE,
  verbose = FALSE,
  ...
)

```

Arguments

<code>model</code>	One <code>likelihood()</code> object or a non-empty list of them.
<code>data</code>	A data frame containing every simple response column, fixed-effect variable, and ordinary latent index. All likelihood blocks use the same number and ordering of rows.
<code>latent.index</code>	Optional <code>index()</code> object or list of such objects.

<code>LKJ.eta</code>	One positive finite LKJ shape shared by every <code>iidkd_LKJ</code> group. The default is 5. This argument is unused if no such term is present.
<code>control.family</code>	Optional outer list containing one INLA <code>control.family</code> list per likelihood block. An available outer element overrides that block's <code>family.setting</code> ; later blocks without an override use their block-level setting. Use the nested outer list even for one block. For LAVM, configure <code>link</code> and <code>hyper\$kappa</code> ; see lavm.cloglike() .
<code>control.fixed</code>	Optional INLA fixed-effect control list. The package defaults are mean 0 and precision 0.001. Named priors from intercept() and covariate() override matching coefficients. Scalar mean/prec values and INLA named lists with a <code>default</code> entry are both supported; existing named entries are preserved.
<code>control.inla</code>	INLA approximation controls. The package default is <code>list(cmin = 0, compute.initial.values = TRUE)</code> . A user-supplied value replaces that complete list rather than being merged with it.
<code>metrics</code>	Logical; if TRUE, request CPO, DIC, WAIC, and INLA configuration output. The default is FALSE.
<code>verbose</code>	Logical passed to <code>INLA::inla()</code> .
<code>...</code>	Further arguments for <code>INLA::inla()</code> . Common choices include <code>control.predictor</code> , <code>control.compute</code> , <code>control.mode</code> , <code>num.threads</code> , and <code>blas.num.threads</code> . A <code>control.compute</code> list is merged field by field with the settings implied by <code>metrics</code> ; other named arguments replace or add call entries.

Details

Each formula response must be a single named column of data. Ordinary numeric right-hand-side terms become block-specific fixed effects. Use [intercept\(\)](#) and [covariate\(\)](#) for stable names or local priors. A bare `f()` call uses the standard R-INLA latent-model syntax and its options are retained in the assembled formula. `INLAcircular` does not export its own `f()`; write `f(...)`, not `INLA::f(...)`, because the assembler recognizes the bare call while preprocessing indices.

A discovered `f()` index is read from data, supplied through `latent.index`, or generated as `seq_len(nrow(data))`. A non-NULL `process.id` is injected as the term's values vector when `var` matches the first `f()` argument, replacing an existing values argument. See [index\(\)](#) for first-, group-, replicate-, and likelihood-index mappings.

A response used directly as a bare term, or as `covariate(response, predictor = NULL)`, on another block's right-hand side is represented by its latent predictor through an INLA copy term. NULL is the default. Set `predictor = TRUE` to request copying explicitly, or `predictor = FALSE` to use the observed response values as an ordinary fixed covariate. A transformed response expression remains an ordinary observed fixed effect; explicitly requesting a copy of one is an error. In models with copied responses, request fitted predictors with `control.predictor = list(compute = TRUE)` when needed.

`model = "iidkd_LKJ"` is a special `graphpcor`-backed extension. Terms with the same bare first index are grouped in likelihood order. The group must occur in at least two blocks and at most once per block. Per-component `pc.prior.u` and `pc.prior.alpha` default to 1 and 0.5; see [LKJcc\(\)](#).

Do not pass formula or family through `...`, because those names replace the internally assembled entries. Configure family and fixed-effect priors through the formal `control.family` and `control.fixed` arguments; the stacked formula and data are owned by `inlacc()`.

Value

The fitted `INLA::inla()` object with class "inlacc" prepended. Raw INLA summaries, marginals, fitted values, criteria, and timing fields are retained. The result also contains `inlacc_meta`, `lkj_modes`, and `inlacc_meta$LKJ`; the latter two are empty lists when no graphpcor-backed effect is used. LAvM concentration marginals are normally transformed from $\log(\kappa)$ to natural-scale κ and renamed.

Author(s)

Xiang Ye <xiang.ye@kaust.edu.sa>

See Also

[likelihood\(\)](#), [intercept\(\)](#), [covariate\(\)](#), [index\(\)](#), [LKJcc\(\)](#), [lavm.cloglike\(\)](#), [summary.inlacc\(\)](#)

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  z <- rnorm(30)
  angle <- rlavm(30, eta = 0.5 + z, kappa = 10)
  angle_model <- likelihood(
    angle ~ intercept(name = "alpha0", mean = 0, sd = 2) +
      covariate(z, name = "alpha_z", mean = 0, sd = 1),
    family = "lavm"
  )

  fit <- inlacc(
    model = angle_model,
    data = data.frame(angle = angle, z = z),
    num.threads = "2:1"
  )
  summary(fit)
}
```

intercept

Define a custom intercept in an inlacc() formula

Description

`intercept()` is a formula marker interpreted by `inlacc()`. It replaces the automatically named `intercept` for that likelihood block.

Usage

```
intercept(name = NULL, mean = NULL, sd = NULL)
```

Arguments

name	Optional coefficient name. The default is <code>Intercept_<response></code> . Use unique, syntactically valid names unless a shared fixed coefficient is intentional.
mean	Optional Gaussian prior mean. If omitted, the corresponding <code>control.fixed</code> mean is inherited.
sd	Optional positive Gaussian prior standard deviation. If omitted, the corresponding <code>control.fixed</code> precision is inherited.

Details

A normal formula intercept (1) creates a block-specific automatic intercept. 0 or -1 removes it. An explicit `intercept()` term suppresses the automatic intercept and adds the custom term, even in a no-intercept formula. The marker is not intended to be evaluated outside a `likelihood()` formula.

Value

NULL, invisibly; `inlacc()` interprets the unevaluated call.

See Also

`likelihood()`, `covariate()`, `inlacc()`

Examples

```
likelihood(
  y ~ intercept(name = "beta0", mean = 0, sd = 5),
  family = "gaussian"
)
```

lavm

The Link-Adjusted von Mises (LAvM) Distribution

Description

Density, distribution function, quantile function, and random generation for the Link-Adjusted von Mises (LAvM) distribution with linear predictor `eta` and concentration parameter `kappa`.

Usage

```
dlavm(x, eta, kappa, log = FALSE)

plavm(q, eta, kappa, link_obj = NULL, log = FALSE, len = 2048L)

qlavm(p, eta, kappa, link_obj = NULL, len = 2048L)

rlavm(n, eta, kappa, link_obj = NULL, len = 2048L)
```

Arguments

x, q	vector of quantiles (angles in radians).
eta	linear predictor vector.
kappa	non-negative concentration parameter.
log	logical; if TRUE, probabilities/densities are given as log.
link_obj	reserved for interface compatibility. The current compiled implementation always uses the inverse-tangent link returned by <code>lavm.link("tan")</code> .
len	integer; resolution of the spline grid used for approximation.
p	vector of probabilities.
n	number of observations. If <code>length(n) > 1</code> , the length is taken to be the number required.

Details

The default transformation from the standard circular space to the linear real line is given by the *inverse tangent link*,

$$g(\cdot) := 2 \arctan(\cdot) \in [-\pi, \pi),$$

resulting in:

$$z = 2 \arctan\left(\tan\left(\frac{x}{2}\right) - \eta\right)$$

The corresponding density function for the Link-Adjusted von Mises (LAvM) distribution is:

$$\pi_{\text{LAvM}}(x \mid \eta, \kappa) = \frac{\exp\left\{\kappa \cos\left(2 \arctan\left(\tan\left(\frac{x}{2}\right) - \eta\right)\right)\right\}}{2\pi I_0(\kappa) \left(1 + \eta^2 - \eta \sin(x) - \eta^2 \sin^2\left(\frac{x}{2}\right)\right)}, \quad x \in (-\pi, \pi)$$

Value

`dlavm` gives the density, `plavm` gives the distribution function, `qlavm` gives the quantile function, and `rlavm` generates random deviates.

Examples

```
# Generate 10 random deviates from the LAvM distribution
# with linear predictor eta = 1.5 and concentration kappa = 3
rlavm(10, eta = 1.5, kappa = 3)

# Calculate the density at specific points
x_grid <- seq(-pi, pi, length.out = 100)
densities <- dlavm(x_grid, eta = 1.5, kappa = 3)

# Plot the density curve
plot(x_grid, densities, type = "l", main = "LAvM Density (eta = 1.5, kappa = 3)",
     xlab = "Angle", ylab = "Density")

# Calculate the cumulative probability up to pi/2
plavm(pi/2, eta = 1.5, kappa = 3)
```

```
# Find the median (50th percentile)
qlavm(0.5, eta = 1.5, kappa = 3)
```

lavm.cloglike

Define the compiled LAVM cloglike for direct R-INLA fitting

Description

Construct the custom likelihood object needed by a literal `INLA::inla(..., family = "cloglike")` call. The convenience `inla()` function and `inlacc()` call this constructor internally.

Usage

```
lavm.cloglike(family.setting = NULL)
```

Arguments

`family.setting` A list of controls for the LAVM likelihood. `link` is "inverse.tangent" (default), "scaled.logit", or "scaled.probit". Within `hyper$kappa`, `initial` is the initial value of $\log(\kappa)$, `prior` is "pc.vminf" (default) or "pc.v θ ", `param = c(u, alpha)` contains two values strictly between zero and one, and `fixed = TRUE` fixes $\log(\kappa)$ at `initial`. For a single likelihood, both `list(link = ..., hyper = ...)` and the nested INLA form `list(list(link = ..., hyper = ...))` are accepted.

Details

The native likelihood entry point is `INLAcirc_cloglike_lavm`. The standalone implementation for `inla-build` is kept in `src-cloglike`; the package DLL uses a self-contained synchronized copy in `src`, so package installation never compiles `src-cloglike`.

In `param = c(u, alpha)`, `u` is a threshold for the mean resultant length $I_1(\kappa) / I_0(\kappa)$, and both supported priors are calibrated so that the probability of exceeding `u` is `alpha`. This is distinct from the standalone PC-prior `d/p/q/r` functions, which accept `lambda` directly. The defaults are `initial = 6`, `prior = "pc.vminf"`, `param = c(0.5, 0.5)`, and `fixed = FALSE`. `hyper$prec` and `log.initial` are rejected.

The standalone `pc.vminf` law has a boundary atom at $\kappa = 0$. INLA represents a finite $\theta = \log(\kappa)$, so this cloglike evaluates the continuous component on the θ scale, including its Jacobian, and has no discrete state at $\theta = -\text{Inf}$. Its `c(u, alpha)` input should therefore be read as a rate calibration. For numerical stability, the compiled likelihood and prior use a working θ restricted to $[-30, 30]$.

For a direct fit, convert the response with `INLA::inla.mdata()`, use `family = "cloglike"`, and pass the returned object as `control.family = list(cloglike = object)`. A literal `INLA::inla(..., family = "lavm")` does not select this package's custom likelihood or priors.

Value

An INLA cloglike object.

See Also

[inla\(\)](#), [inlacc\(\)](#)

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  z <- seq(-1, 1, length.out = 30)
  y <- rlavm(length(z), eta = 0.5 + z, kappa = 50)
  y_lavm <- INLA::inla.mdata(y)
  cloglike <- lavm.cloglike()
  fit <- INLA::inla(
    y_lavm ~ z,
    family = "cloglike",
    data = list(y_lavm = y_lavm, z = z),
    control.family = list(cloglike = cloglike),
    num.threads = "2:1"
  )
  fit$summary.fixed
}
```

lavm.link

Create Link-Adjusted von Mises (LAvM) Link Object

Description

Create Link-Adjusted von Mises (LAvM) Link Object

Usage

```
lavm.link(type = "tan")
```

Arguments

type Character string defining the link function. Default is "tan".

Value

A list of class lavm_link containing the transform and inverse functions.

likelihood	<i>Define one likelihood block for <code>inlacc()</code></i>
------------	--

Description

A likelihood block stores one response formula, its likelihood family, and the corresponding INLA family controls. Pass one block, or a list of blocks, to `inlacc()`.

Usage

```
likelihood(formula, family, family.setting = NULL)
```

Arguments

formula	A two-sided model formula defining one response block. The left side should be a single column name in data; the right side may contain numeric fixed-effect expressions, <code>intercept()</code> , <code>covariate()</code> , and bare INLA <code>f()</code> terms.
family	One character string naming the likelihood family. Use "lavm" for the package's link-adjusted von Mises likelihood. Other family names are passed to <code>INLA::inla()</code> .
family.setting	Optional per-block INLA control.family list. For an LAvM block, use link and hyper\$kappa; see <code>lavm.cloglike()</code> .

Details

All blocks supplied to one `inlacc()` call share the rows of a single data frame. Their order determines the stacking order and, for an iidkd_LKJ group, the multivariate component order. This constructor stores the specification; validation that depends on the data occurs in `inlacc()`.

Value

An object of class `inlacc_model` containing formula, family, and family.setting.

See Also

`inlacc()`, `intercept()`, `covariate()`, `index()`

Examples

```
block <- likelihood(
  angle ~ intercept(name = "alpha0", mean = 0, sd = 2) +
    covariate(z, name = "alpha_z", mean = 0, sd = 1),
  family = "lavm",
  family.setting = list(
    link = "inverse.tangent",
    hyper = list(
      kappa = list(
        initial = 6,
```

```

      prior = "pc.vminf",
      param = c(0.5, 0.5),
      fixed = FALSE
    )
  )
)

```

LKJcc

Prepare graphpcor LKJ covariance components for inlacc()

Description

LKJcc() finds every `f(..., model = "iidkd_LKJ")` term, groups terms that use the same index name, constructs one graphpcor LKJ model per group, rewrites the formulas, and creates the component and replicate indices required by INLA.

Usage

```
LKJcc(formulas, LKJ.eta = 5, n.obs = NULL, latent.index = NULL)
```

Arguments

<code>formulas</code>	One formula or a non-empty list of formulas, in likelihood order.
<code>LKJ.eta</code>	One positive finite LKJ shape shared by all detected groups. The default is 5.
<code>n.obs</code>	Positive integer number of observations in each likelihood block. It is required when an LKJ term is present.
<code>latent.index</code>	Optional <code>index()</code> specification or list. For an LKJ index, <code>process.id = "likelihood"</code> means that <code>data.id</code> supplies the replicate IDs and the LKJ component index is generated from likelihood order.

Details

In every `f(index, model = "iidkd_LKJ")` term, `pc.prior.u` defaults to 1 and `pc.prior.alpha` defaults to 0.5. They are forwarded to graphpcor as `sigma.prior.reference` and `sigma.prior.probability`. The reference must be positive; the probability may be in $[0, 1]$ or NA. Values may differ by likelihood component. For an estimated scale, the implementation calibrates the PC prior by $P(\text{sigma} > \text{pc.prior.u}) = \text{pc.prior.alpha}$, equivalently with rate $-\log(\text{pc.prior.alpha}) / \text{pc.prior.u}$. A probability of 0, 1, or NA fixes that component's standard deviation at `pc.prior.u`. Supply the common LKJ.eta through `inlacc()`, not as `eta` inside `f()`; `prior.param` is not supported.

A group is identified by the bare first argument of `f()` and component order follows formula order. It must occur in at least two formulas, at most once per formula, and its index name cannot also identify another `f()` model. The exact character value `model = "iidkd_LKJ"` is required.

Replicate IDs default to `seq_len(n.obs)`. To supply them, use `index(var = "i", data.id = replicate_ids, process.id = "likelihood")`, where every replicate ID is a positive integer. An explicit `replicate = name` inside the special `f()` controls the generated column name, not its

values. It must be a bare name, and all explicit replicate names within one group must agree; a name supplied in one component is propagated to the whole group.

Actual construction requires both the suggested graphpcor and INLAtools packages. This function supports the dense LKJ-correlated IID model only; it is not a general interface to graphpcor's graph or tree model families. It is normally called by `inlacc()` rather than directly.

Value

A list with components `formulas`, `models`, `block.data`, `lkj_vars`, `replicate_vars`, `likelihoods`, `processes`, `data.vars`, and `latent.index`. The last component contains mappings not consumed by the LKJ preprocessor.

See Also

`inlacc()`, `index()`

Examples

```
if (requireNamespace("graphpcor", quietly = TRUE) &&
    requireNamespace("INLAtools", quietly = TRUE)) {
  # The LKJ model also requires graphpcor's compiled shared library.
  # Its path lookup warns when that optional library is absent.
  graphpcor_lib <- suppressWarnings(INLAtools::cgeneric_shlib_path(
    package = "graphpcor", useINLApcomp = FALSE
  ))
  if (file.exists(graphpcor_lib)) {
    formulas <- list(
      y1 ~ f(i, model = "iidkd_LKJ"),
      y2 ~ f(i, model = "iidkd_LKJ")
    )
    prepared <- LKJcc(formulas, LKJ.eta = 5, n.obs = 5)
    prepared$formulas
    prepared$block.data
  }
}
```

pc_card

PC Prior for Cardioid Concentration: Cardioid Base

Description

Density, distribution function, quantile function, and random generation for the penalized-complexity prior on cardioid concentration with the maximally concentrated cardioid model ($\kappa = 1/2$) as its base.

Usage

```
dpc.card(kappa, lambda, log = FALSE)

ppc.card(q, lambda, log.p = FALSE)

qpc.card(p, lambda, log.p = FALSE)

rpc.card(n, lambda)
```

Arguments

kappa, q	Vectors of cardioid concentration values.
lambda	Positive rate parameter on the PC-distance scale.
log	Logical scalar; if TRUE, return log-densities.
log.p	Logical scalar; if TRUE, probabilities are supplied or returned on the log scale.
p	Vector of probabilities.
n	Number of observations. If <code>length(n) > 1</code> , its length is used.

Details

For $0 \leq \kappa \leq 1/2$, let

$$s(\kappa) = \sqrt{1 - 4\kappa^2}$$

and

$$d_{1/2}(\kappa) = \sqrt{1 - 2\kappa - s(\kappa) + \log(1 + s(\kappa))}.$$

This distance decreases from $D_{1/2} = \sqrt{\log 2}$ at zero to zero at the base model. The continuous density and CDF are

$$\pi_{1/2}(\kappa) = \lambda e^{-\lambda d_{1/2}(\kappa)} \frac{1 + s(\kappa) - 2\kappa}{(1 + s(\kappa))d_{1/2}(\kappa)},$$

$$F_{1/2}(\kappa) = e^{-\lambda d_{1/2}(\kappa)}.$$

Since $F_{1/2}(0) = e^{-\lambda\sqrt{\log 2}}$, the distribution has a boundary atom of that size at $\kappa = 0$. `dpc.card` returns the continuous density only; `ppc.card`, `qpc.card`, and `rpc.card` include the atom. The continuous density has an integrable singularity at $\kappa = 0.5$, where it is returned as `Inf`. At zero, the continuous density is $\lambda \exp\{-\lambda\sqrt{\log 2}\}/\sqrt{\log 2}$; this is separate from the atom probability.

All endpoint series, log-scale calculations, inversion, and random generation are implemented in compiled C code.

Value

`dpc.card` gives the continuous density, `ppc.card` gives the distribution function, `qpc.card` gives the quantile function, and `rpc.card` generates random deviates.

Examples

```
dpc.card(kappa = c(0, 0.25, 0.49), lambda = 2)
ppc.card(q = 0.25, lambda = 2)
qpc.card(p = 0.5, lambda = 2)
rpc.card(n = 10, lambda = 2)
```

pc_card0

PC Prior for Cardioid Concentration: Uniform Base

Description

Density, distribution function, quantile function, and random generation for the penalized-complexity prior on cardioid concentration with the circular uniform distribution ($\kappa = 0$) as its base model.

Usage

```
dpc.card0(kappa, lambda, log = FALSE)

ppc.card0(q, lambda, log.p = FALSE)

qpc.card0(p, lambda, log.p = FALSE)

rpc.card0(n, lambda)
```

Arguments

kappa, q	Vectors of cardioid concentration values.
lambda	Positive rate parameter on the PC-distance scale.
log	Logical scalar; if TRUE, return log-densities.
log.p	Logical scalar; if TRUE, probabilities are supplied or returned on the log scale.
p	Vector of probabilities.
n	Number of observations. If $\text{length}(n) > 1$, its length is used.

Details

For $0 \leq \kappa \leq 1/2$, let

$$s(\kappa) = \sqrt{1 - 4\kappa^2}$$

and

$$d_0(\kappa) = \sqrt{1 - s(\kappa) + \log\{(1 + s(\kappa))/2\}}.$$

The largest attainable distance is $D_0 = \sqrt{1 - \log 2}$. Consequently, the exponential PC prior on distance is truncated and normalized on $[0, D_0]$:

$$\pi_0(\kappa) = \frac{\lambda e^{-\lambda d_0(\kappa)}}{1 - e^{-\lambda D_0}} \frac{2\kappa}{(1 + s(\kappa))d_0(\kappa)},$$

$$F_0(\kappa) = \frac{1 - e^{-\lambda d_0(\kappa)}}{1 - e^{-\lambda D_0}}.$$

The endpoint densities are $\lambda/\{1 - \exp(-\lambda D_0)\}$ at zero and $\lambda \exp(-\lambda D_0)/\{D_0[1 - \exp(-\lambda D_0)]\}$ at $\kappa = 1/2$. Algebraic reductions, small-kappa series, $\log1p/\expm1$ calculations, numerical inversion, and random generation are all performed in compiled C code.

Value

`dpc.card0` gives the density, `ppc.card0` gives the distribution function, `qpc.card0` gives the quantile function, and `rpc.card0` generates random deviates.

Examples

```
dpc.card0(kappa = c(0, 0.25, 0.5), lambda = 2)
ppc.card0(q = 0.25, lambda = 2)
qpc.card0(p = 0.5, lambda = 2)
rpc.card0(n = 10, lambda = 2)
```

pc_vm0

PC Prior for von Mises Concentration: Uniform Base

Description

Density, distribution function, quantile function, and random generation for the penalized-complexity prior on the von Mises concentration parameter with the circular uniform distribution ($\kappa = 0$) as its base model.

Usage

```
dpc.vm0(kappa, lambda, log = FALSE)
ppc.vm0(q, lambda, log.p = FALSE)
qpc.vm0(p, lambda, log.p = FALSE)
rpc.vm0(n, lambda)
```

Arguments

<code>kappa, q</code>	Vector of non-negative concentration values.
<code>lambda</code>	Positive rate parameter.
<code>log</code>	Logical; if TRUE, return the log-density.
<code>log.p</code>	Logical; if TRUE, probabilities are supplied or returned on the log scale.
<code>p</code>	Vector of probabilities.
<code>n</code>	Number of observations. If <code>length(n) > 1</code> , its length is used.

Details

The PC distance from the uniform base model is

$$d_0(\kappa) = \sqrt{\kappa I_1(\kappa)/I_0(\kappa) - \log I_0(\kappa)}.$$

Write $A(\kappa) = I_1(\kappa)/I_0(\kappa)$ and $A'(\kappa) = 1 - A(\kappa)^2 - A(\kappa)/\kappa$, with the continuous limit $A'(0) = 1/2$. The density and CDF are

$$\pi_0(\kappa) = \lambda \exp\{-\lambda d_0(\kappa)\} \frac{\kappa A'(\kappa)}{2d_0(\kappa)},$$

$$F_0(\kappa) = 1 - \exp\{-\lambda d_0(\kappa)\}.$$

The density has the finite endpoint limit $\pi_0(0) = \lambda/2$; there is no atom at zero.

All Bessel evaluations, small- κ density expansions, large- κ asymptotic expansions, CDF calculations, and numerical inversion are performed by the package's compiled C code.

Value

`dpc.vminf` gives the density, `ppc.vminf` gives the distribution function, `qpc.vminf` gives the quantile function, and `rpc.vminf` generates random deviates.

Examples

```
dpc.vminf(kappa = c(0, 1, 5), lambda = 2)
ppc.vminf(q = 5, lambda = 2)
qpc.vminf(p = 0.5, lambda = 2)
rpc.vminf(n = 10, lambda = 2)
```

pc_vminf

PC Prior for von Mises Concentration: Point-Mass Base

Description

Density, distribution function, quantile function, and random generation for the penalized-complexity prior on the von Mises concentration parameter with the point-mass limit ($\kappa \rightarrow \infty$) as its base model.

Usage

```
dpc.vminf(kappa, lambda, log = FALSE)

ppc.vminf(q, lambda, log.p = FALSE)

qpc.vminf(p, lambda, log.p = FALSE)

rpc.vminf(n, lambda)
```

Arguments

kappa, q	Vector of non-negative concentration values.
lambda	Positive rate parameter.
log	Logical; if TRUE, return the log-density.
log.p	Logical; if TRUE, probabilities are supplied or returned on the log scale.
p	Vector of probabilities.
n	Number of observations. If length(n) > 1, its length is used.

Details

The PC distance from the point-mass base model is

$$d_{\infty}(\kappa) = \sqrt{1 - I_1(\kappa)/I_0(\kappa)}.$$

Writing $A(\kappa) = I_1(\kappa)/I_0(\kappa)$ and $A'(\kappa) = 1 - A(\kappa)^2 - A(\kappa)/\kappa$, the continuous density and full CDF are

$$\pi_{\infty}^{\text{cont}}(\kappa) = \lambda \exp\{-\lambda d_{\infty}(\kappa)\} \frac{A'(\kappa)}{2d_{\infty}(\kappa)},$$

$$F_{\infty}(\kappa) = \exp\{-\lambda d_{\infty}(\kappa)\}.$$

Because $d_{\infty}(0) = 1$, the CDF contains boundary mass $\exp(-\lambda)$ at $\kappa = 0$. The continuous density has right-hand limit $\lambda \exp(-\lambda)/4$ there. `dpc.vminf` returns the continuous density component; `ppc.vminf`, `qpc.vminf`, and `rpc.vminf` include the boundary mass. All numerical calculations, including the large- κ asymptotic branch and quantile inversion, are performed by compiled C code.

Value

`dpc.vminf` gives the continuous density, `ppc.vminf` gives the distribution function, `qpc.vminf` gives the quantile function, and `rpc.vminf` generates random deviates.

Examples

```
dpc.vminf(kappa = c(0, 1, 5), lambda = 2)
ppc.vminf(q = 5, lambda = 2)
qpc.vminf(p = 0.5, lambda = 2)
rpc.vminf(n = 10, lambda = 2)
```

pc_wc

*PC Prior for Wrapped Cauchy Concentration***Description**

Density, distribution function, quantile function, and random generation for the penalized-complexity prior on wrapped Cauchy concentration with the circular uniform distribution ($\kappa = 0$) as its base model.

Usage

```
dpc.wc(kappa, lambda, log = FALSE)
```

```
ppc.wc(q, lambda, log.p = FALSE)
```

```
qpc.wc(p, lambda, log.p = FALSE)
```

```
rpc.wc(n, lambda)
```

Arguments

kappa, q	Vectors of wrapped Cauchy concentration values.
lambda	Positive rate parameter on the PC-distance scale.
log	Logical scalar; if TRUE, return log-densities.
log.p	Logical scalar; if TRUE, probabilities are supplied or returned on the log scale.
p	Vector of probabilities.
n	Number of observations. If length(n) > 1, its length is used.

Details

For $0 \leq \kappa < 1$, the PC distance is

$$d_{wc}(\kappa) = \sqrt{-\log(1 - \kappa^2)}.$$

It maps the parameter support to the full non-negative distance axis, so the density and CDF require no truncation:

$$\pi_{wc}(\kappa) = \lambda e^{-\lambda d_{wc}(\kappa)} \frac{\kappa}{(1 - \kappa^2) d_{wc}(\kappa)},$$

$$F_{wc}(\kappa) = 1 - e^{-\lambda d_{wc}(\kappa)}.$$

The density limit at zero is λ ; it diverges as kappa approaches one and is returned as Inf at that limiting endpoint. The analytic quantile is evaluated with stable log1p/expm1 calculations. Interior-probability quantiles ($0 < p < 1$) that would round to exactly one are kept at the largest representable value below one.

All density, CDF, quantile, and random-generation calculations are performed in compiled C code.

Value

dpc.wc gives the density, ppc.wc gives the distribution function, qpc.wc gives the quantile function, and rpc.wc generates random deviates.

Examples

```
dpc.wc(kappa = c(0, 0.5, 0.9), lambda = 2)
ppc.wc(q = 0.5, lambda = 2)
qpc.wc(p = 0.5, lambda = 2)
rpc.wc(n = 10, lambda = 2)
```

```
print.inlacc_summary    Print an inlacc summary
```

Description

Print timing, posterior summaries, prior labels, family or latent-process labels, and optionally the available model-comparison criteria.

Usage

```
## S3 method for class 'inlacc_summary'
print(x, decimal = NULL, metrics = TRUE, ...)
```

Arguments

x	An inlacc_summary object returned by summary.inlacc() .
decimal	Optional number of decimal places to print. If NULL, use x\$decimal, falling back to 3.
metrics	Logical; whether to print available marginal log-likelihood, DIC, and WAIC values. This cannot create criteria that were not requested when fitting the model.
...	Additional arguments, currently unused.

Details

The printed table combines likelihood, fixed-effect, and random effect summaries while retaining section labels. A final note recommends `control.predictor = list(compute = TRUE)` for models where fitted predictor marginals are particularly relevant and were not requested.

Value

x, invisibly.

See Also

[inlacc\(\)](#), [summary.inlacc\(\)](#), [print.inlacc_time\(\)](#)

Examples

```

if (requireNamespace("INLA", quietly = TRUE)) {
  dat <- data.frame(y = c(0.7, 1.1, 0.9, 1.4, 1.2, 0.8))
  model <- likelihood(y ~ 1, family = "gaussian")
  fit <- inlacc(model, data = dat, num.threads = "2:1")
  fit_summary <- summary(fit, decimal = 4)
  print(fit_summary, decimal = 2, metrics = FALSE)
}

```

print.inlacc_time	<i>Print elapsed times from an inlacc() fit</i>
-------------------	---

Description

Format the setup, INLA computation, post-processing, and total elapsed times stored by an `inlacc()` result.

Usage

```

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

```

Arguments

<code>x</code>	A named numeric <code>inlacc_time</code> object with entries Pre, Running, Post, and Total, measured in seconds.
<code>...</code>	Additional arguments, currently unused.

Value

`x`, invisibly.

See Also

`inlacc()`, `summary.inlacc()`, `print.inlacc_summary()`

Examples

```

timing <- structure(
  c(Pre = 0.1, Running = 1.2, Post = 0.2, Total = 1.5),
  class = c("inlacc_time", "numeric")
)
print(timing)

```

summary.inlacc	<i>Summarize an inlacc() fit</i>
----------------	----------------------------------

Description

Transform and organize the fixed effects, likelihood parameters, latent hyperparameters, timing, and requested model criteria from an [inlacc\(\)](#) result.

Usage

```
## S3 method for class 'inlacc'
summary(object, decimal = 3L, ...)
```

Arguments

object	A fitted inlacc model.
decimal	Number of decimal places used later by the print method. It does not round values stored in the returned object.
...	Additional arguments, currently unused.

Value

An object of class `inlacc_summary`, which is a list containing:

- `fixed_par`: fixed-effect posterior summaries and displayed Gaussian priors;
- `likelihood_par`: transformed likelihood-parameter summaries, or `NULL`;
- `random_par`: transformed latent-process hyperparameter summaries, or `NULL`;
- `time.summary`: a named `inlacc_time` vector in seconds;
- `metrics`: available marginal log-likelihood, DIC, and WAIC entries;
- `needs_pred_warning`: whether predictor computation is recommended; and
- `decimal`: the requested printing precision.

Posterior summary tables use columns mean, SD, 2.5%, 50%, 97.5%, mode, and prior; likelihood and random-effect tables also carry family or process-type information when available.

See Also

[inlacc\(\)](#), [print.inlacc_summary\(\)](#), [print.inlacc_time\(\)](#)

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  dat <- data.frame(y = c(0.7, 1.1, 0.9, 1.4, 1.2, 0.8))
  model <- likelihood(y ~ 1, family = "gaussian")
  fit <- inlacc(model, data = dat, num.threads = "2:1")
  fit_summary <- summary(fit, decimal = 4)
  fit_summary$fixed_par
  print(fit_summary, metrics = FALSE)
}
```

von_mises

The von Mises Distribution

Description

Density, distribution function, quantile function, and random generation for the von Mises distribution with mean direction μ and concentration parameter κ .

Usage

```
dvm(x, mu, kappa, log = FALSE)
```

```
pvm(q, mu, kappa, strategy = "circular", log = FALSE, len = 2048L)
```

```
qvm(p, mu, kappa, len = 2048L)
```

```
rvm(n, mu, kappa, len = 2048L)
```

Arguments

x, q	vector of quantiles (angles in radians).
mu	mean direction of the distribution (in radians).
kappa	non-negative concentration parameter.
log	logical; if TRUE, probabilities/densities are given as log.
strategy	character; either "circular" or "linear" for the CDF integration strategy. If "circular", the CDF integration starts from $\mu - \pi$. If "linear", it starts from $-\pi$.
len	integer; resolution of the spline grid used for approximation.
p	vector of probabilities.
n	number of observations. If $\text{length}(n) > 1$, the length is taken to be the number required.

Details

The von Mises distribution has the following density function:

$$f(x | \mu, \kappa) = \frac{1}{2\pi I_0(\kappa)} \exp\{\kappa \cos(x - \mu)\}$$

where $I_0(\kappa)$ is the modified Bessel function of the first kind of order 0.

Value

dvm gives the density, pvm gives the distribution function, qvm gives the quantile function, and rvm generates random deviates.

Examples

```
# Generate 10 random deviates from a von Mises distribution
# with mean direction \eqn{\pi/2} and concentration 5
rvm(10, mu = pi/2, kappa = 5)

# Calculate the density at specific points
x_grid <- seq(-pi, pi, length.out = 100)
densities <- dvm(x_grid, mu = 0, kappa = 2)

# Plot the density curve
plot(x_grid, densities, type = "l", main = "von Mises Density (kappa = 2)",
      xlab = "Angle", ylab = "Density")

# Calculate the cumulative probability up to pi/2
pvm(pi/2, mu = 0, kappa = 2)

# Find the 95th percentile
qvm(0.95, mu = 0, kappa = 2)
```

wind_newyork

New York hourly wind climate normals

Description

A processed full-year dataset of hourly wind direction, wind speed, and temperature normals for JFK International Airport, New York. The calendar variables are derived from DATE, and wind direction is stored in radians as a `circular::circular` vector.

Format

A data frame with 8,759 rows and 8 variables:

DATE Character timestamp in MM-DDTHH:MM:SS format.

HLY.WIND.VCTDIR Circular wind direction in radians.

HLY.WIND.VCTSPD Numeric vector wind speed.

HLY.TEMP.NORMAL Numeric vector normal temperature.

datetime POSIXct timestamp in UTC, using 2010 as the reference year.

month Integer month.

day Integer day of the month.

hour Integer hour on the 0–23 scale.

Source

JFK International Airport station USW00094789; source data supplied with the New York wind application.

Examples

```
data(wind_newyork)
dim(wind_newyork)
class(wind_newyork$HLY.WIND.VCTDIR)
range(wind_newyork$month)
```

Index

* **datasets**
 wind_newyork, 29

bessel_i, 2

circular::circular, 29
circular_densities, 3
covariate, 5
covariate(), 10–12, 16

dcard(circular_densities), 3
dcardioid(circular_densities), 3
dlavm(lavm), 12
dpc.card(pc_card), 18
dpc.card0(pc_card0), 20
dpc.vm0(pc_vm0), 21
dpc.vminf(pc_vminf), 22
dpc.wc(pc_wc), 24
dvm(von_mises), 28
dwc(circular_densities), 3
dwrappedcauchy(circular_densities), 3

index, 6
index(), 9–11, 16–18
inla, 7
inla(), 14, 15
inlacc, 9
inlacc(), 5–8, 11, 12, 14–18, 25–27
intercept, 11
intercept(), 6, 10, 11, 16

lavm, 12
lavm.cloglike, 14
lavm.cloglike(), 8, 10, 11, 16
lavm.link, 15
likelihood, 16
likelihood(), 6, 9, 11, 12
LKJcc, 17
LKJcc(), 7, 10, 11

pc.card(pc_card), 18
pc.card0(pc_card0), 20
pc.vm0(pc_vm0), 21
pc.vminf(pc_vminf), 22
pc.wc(pc_wc), 24
pc_card, 18
pc_card0, 20
pc_vm0, 21
pc_vminf, 22
pc_wc, 24
plavm(lavm), 12
ppc.card(pc_card), 18
ppc.card0(pc_card0), 20
ppc.vm0(pc_vm0), 21
ppc.vminf(pc_vminf), 22
ppc.wc(pc_wc), 24
print.inlacc_summary, 25
print.inlacc_summary(), 26, 27
print.inlacc_time, 26
print.inlacc_time(), 25, 27
pvm(von_mises), 28

qlavm(lavm), 12
qpc.card(pc_card), 18
qpc.card0(pc_card0), 20
qpc.vm0(pc_vm0), 21
qpc.vminf(pc_vminf), 22
qpc.wc(pc_wc), 24
qvm(von_mises), 28

rlavm(lavm), 12
rpc.card(pc_card), 18
rpc.card0(pc_card0), 20
rpc.vm0(pc_vm0), 21
rpc.vminf(pc_vminf), 22
rpc.wc(pc_wc), 24
rvm(von_mises), 28

summary.inlacc, 27
summary.inlacc(), 11, 25, 26

von_mises, 28

wind_newyork, [29](#)