Package: BiFactorCopulaGAMM (via r-universe)

November 14, 2026

Type Package

Title Bi-Factor Copula-Based Mixed Regression Models

Version 0.1.9

Description Estimation of 2-level bi-factor copula regression models for clustered data where the (multivariate) response variables can be either discrete or continuous, while the copulas for the bi-factor models are taken from the package 'CopulaGAMM'.

Depends R (>= 3.5.0), CopulaGAMM

License GPL (>= 2)

Encoding UTF-8

Imports stats, statmod

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Pavel Krupskii [aut, ctb, cph], Bouchra R. Nasri [aut, ctb, cph], Bruno N Remillard [aut, cre, cph]

Maintainer Bruno N Remillard <bruno.remillard@hec.ca>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-13 23:10:21 UTC

RemoteUrl https://github.com/cran/BiFactorCopulaGAMM

RemoteRef HEAD

RemoteSha 164971aa15367e4e4bfa20be28cb37699b248fd2

Contents

bounds.cop 2
bounds.mar 3
condMeansBifactor 4
condMeanV_bifactor 5
condMeanY 6

condMeanY_given_X . . . 7
condMeanY_given_XV . . . 8
condMediansBifactor . . . 9
condMedianV_bifactor . . . 10
cpar2link . . . 11
dlap . . . 12
EstBifactorCopulaGAMMFast . . . 12
EstBifactorTwoStep . . . 15
EstMargin1Factor . . . 17
expcpdf . . . 19
geomcpdf . . . 19
invfuncNew . . . 20
lapcpdf . . . 21
link2parM . . . 21
margins0 . . . 22
nbinom1cpdf . . . 23
nbinomcpdf . . . 23
normcpdf . . . 24
param.dim . . . 24
plap . . . 25
poiscpdf . . . 25
qcondcla . . . 26
qcondfgm . . . 26
qcondfra . . . 27
qcondgal . . . 27
qcondgum . . . 28
qcondhr . . . 28
qcondjoe . . . 29
qcondNew . . . 29
qcondnor . . . 30
qcondpla . . . 31
qcondt . . . 31
qlap . . . 32
qmargin . . . 32
Sim2Cluster . . . 33
tcpdf . . . 34
weibcpdf . . . 35

Index **36**

---

bounds.cop *Starting values, upper and lower bounds for parameter estimation of bi-factor copulas*

Description

The starting point is a value giving approximately a Kendall's tau of 0.5, and the upper and lower bounds are values giving a tau of 0.99 (resp. 0.1 or -0.99) for the copula families

Usage

```
bounds.cop(cop)
```

Arguments

cop	Vector of copula families: "gaussian" (normal), "t" , "clayton" , "frank" , "gumbel", "galambos", "joe", "huesler-reiss", "plackett".

Value

out List of starting points (start), upper bounds (UB) and lower bounds (LB)

Examples

```
cop=c("gumbel","gumbel","normal","clayton","frank","clayton")
out=bounds.cop
```

---

bounds.mar *Starting values, upper and lower bounds for parameter estimation of margins*

Description

The starting point is an estimation of the parameters and the upper and lower bounds are approximate 99.7

Usage

```
bounds.mar(X, margin)
```

Arguments

X	Matrix of data
margin	Vector of margin families: "binomial" (bernoulli), "poisson", "exponential", "normal"..

Value

out List of starting points (start), upper bounds (UB) and lower bounds (LB)

Examples

```
margin=c("bernoulli","poisson","normal")
U=matrix(runif(300),ncol=3)
X = cbind(as.double(U[,1]>.3),qpois(U[,2],5),10+2*qnorm(U[,3]))
out=bounds.mar(X,margin)
```

`condMeansBifactor` *Conditional mean of V for every cluster, all Y/X components jointly*

Description

Same interface/bookkeeping as ‘condMediansBifactor()‘. When the batched ‘cond_median_logh_multi()‘ kernel is available, computes every cluster’s mean from the SAME single, all-K-clusters-at-once evaluation ‘.prepare_bifactor_batch()‘ performs (the mean needs no bisection, so this is just one weighted average per cluster after that one shared call) – otherwise falls back to the original per-cluster loop over ‘condMeanV_bifactor()‘.

Usage

```
condMeansBifactor(
  Y,
  X = NULL,
  clu,
  model,
  coef,
  nq = 31,
  use_c = TRUE,
  kernel_path = "bifactor_kernel.c"
)
```

Arguments

`Y, X`	N x d / N x p matrices (or vectors) of response/covariate observations; X may be NULL
`clu`	cluster membership, length N
`model`	same model spec as ‘Sim2Cluster()‘/‘EstBifactorTwoStep()‘
`coef`	the fitted ‘coef‘ list (‘$margins‘, ‘$copulas‘) – see the ASSUMPTIONS note at the top of this file re: natural units
`nq`	number of quadrature nodes (default 31)
`use_c`	logical (default TRUE), tries the compiled kernel(s) (batched multi-cluster first, then single-cluster), falling back to pure R
`kernel_path`	path to ‘bifactor_kernel.c‘/‘.so‘

Value

data.frame with one row per cluster: ‘cluster‘, ‘V‘ (conditional posterior mean)

`condMeanV_bifactor` *Conditional (posterior) MEAN of the cluster-level factor V, for ONE cluster, all margins jointly*

Description

Companion to 'condMedianV_bifactor()', computing the conditional posterior MEAN of V_k instead of its median: 'E[V_k | data] = integral v * h_k(v) dv / integral h_k(v) dv'. This is substantially cheaper than the median, since it only needs 'h_k(v)' evaluated ONCE, at the base quadrature grid – unlike the median, which needs 'invfunc()''s ~40 bisection evaluations, each at a different rescaled grid. Reuses the exact same '.lden_grid_bifactor()' machinery (and, when available, the compiled 'cond_median_logh()' C kernel) as 'condMedianV_bifactor()' – source that part of the file first.

Usage

```
condMeanV_bifactor(
  F1,
  F2 = NULL,
  cont,
  fam1,
  rot1,
  cpar1,
  dfC1 = NULL,
  fam2,
  rot2,
  cpar2,
  dfC2 = NULL,
  nq = 31,
  gl = NULL,
  use_c = TRUE,
  kernel_path = "bifactor_kernel.c"
)
```

Arguments

`F1`	n_k x dp matrix: F_j(x_ij) (or G_j(y_ij) for Y-margins) for this cluster's members (columns = margins, Y columns first then X, matching 'model''s order)
`F2`	n_k x dp matrix: the left-limit CDF, needed only for discrete margins (columns for continuous margins can be NA)
`cont`	logical, length dp: TRUE = continuous (uses copula density), FALSE = discrete (uses copula CDF difference)
`fam1`, `rot1`, `cpar1`, `dfC1`	length-dp vectors: copula-1 (links each margin to V) family/rotation/parameter/df
`fam2`, `rot2`, `cpar2`, `dfC2`	length-dp vectors: copula-2 (links each margin's V-conditional value to the per-observation nuisance w) family/rotation/parameter/df

nq	number of Gauss-Legendre-on-(0,1) quadrature nodes, used for BOTH the v-integral and each member's own w-integral (default 31, matching MAP.continuous()/MAP.discrete())
gl	optional precomputed 'statmod::gauss.quad.prob(nq)' result, to avoid recomputing it once per cluster ('condMediansBifactor()' supplies this)
use_c	logical (default TRUE). If TRUE, tries to use the compiled 'cond_median_logh()' C kernel (see 'kernel_path') for the expensive nested member/margin/w-node computation, falling back to the pure-R version automatically if the kernel can't be found/compiled/loaded. Set to FALSE to force the pure-R path (e.g. for debugging).
kernel_path	path to 'bifactor_kernel.c' (or its already-compiled '.so'/'.dll') – the SAME file 'EstBifactorCopulaGAMMFast()' uses, since 'cond_median_logh()' was added alongside its existing 'bifactor_negloglik()'. Defaults to '"bifactor_kernel.c"' in the working directory.

Value

scalar: the conditional posterior mean of V_k for this cluster

---

condMeanY *Conditional mean of every Y-component given X, with or without a predicted V*

Description

Single entry point combining 'condMeanY_given_XV()' and 'condMeanY_given_X()': if 'V' is supplied, computes E[Y_j|X=x,V=v] (eqs. 2.5/2.6), plugging in each row's cluster's predicted latent factor; if 'V=NULL' (default), computes E[Y_j|X=x] instead (eqs. 2.10/2.11), integrating the latent factor out entirely – in that case 'clu' isn't needed either (silently ignored if supplied).

Usage

```
condMeanY(
  Y = NULL,
  X = NULL,
  clu = NULL,
  model,
  coef,
  V = NULL,
  newdata = NULL,
  nq = NULL
)
```

Arguments

Y, X	original training data; only used (and only needed) when 'newdata=NULL'
clu	original training cluster membership; only used when 'newdata=NULL' AND 'V' is supplied
model	same model spec as 'Sim2Cluster()'/'EstBifactorTwoStep()'
coef	the fitted 'coef' list ('$margins', '$copulas')
V	NULL (default), or a data.frame with 'cluster'/'V' columns (as returned by 'condMeansBifactor()'/'condMediansBifactor()') to condition on – see 'condMeanY_given_XV()'
newdata	NULL (default) to re-predict the training data, or a list with 'x' (and, when 'V' is supplied, 'clu') – see the two underlying functions
nq	number of quadrature nodes. Defaults to 31 when 'V' is supplied ('condMeanY_given_XV()''s default) or 25 when it isn't ('condMeanY_given_X()''s default, since that path needs one more nested integral); pass a value explicitly to override either.

Value

data.frame, one row per observation, columns 'EY1',...,'EYd'

---

condMeanY_given_X *Conditional mean of every Y-component given X only (V integrated out), for each observation*

Description

E[Y_j | X=x] for j=1..d, for every observation, using each observation's OWN X covariates and integrating the cluster-level latent factor V out entirely – no predicted V needed at all (and hence no 'clu'/'Vhat' either). Eq. (2.10) for continuous Y_j, eq. (2.11) for discrete Y_j.

Usage

```
condMeanY_given_X(X = NULL, model, coef, newdata = NULL, nq = 25)
```

Arguments

X	original training covariates; only used (and only needed) when 'newdata=NULL'
model	same model spec as 'Sim2Cluster()'/'EstBifactorTwoStep()'
coef	the fitted 'coef' list ('$margins', '$copulas')
newdata	NULL (default) to re-predict the training X's, or a list with 'x' (new X covariates, matrix/vector)
nq	number of quadrature nodes for v, w, AND the Y-quantile grid (default 25 – this function needs one more nested integral than 'condMeanY_given_XV()', so a smaller default keeps it from being needlessly slow; raise it if precision looks off)

Value

data.frame, one row per observation, columns ‘EY1‘,...,‘EYd‘

`condMeanY_given_XV` *Conditional mean of every Y-component given X and a predicted V, for each observation*

Description

E[Y_j | X=x, V=v] for j=1..d, for every observation, using each observation's OWN X covariates and its CLUSTER's predicted latent factor V (from ‘condMeansBifactor()‘/‘condMediansBifactor()‘) – eq. (2.5) for continuous Y_j, eq. (2.6) for discrete Y_j.

Usage

```
condMeanY_given_XV(
  Y = NULL,
  X = NULL,
  clu = NULL,
  model,
  coef,
  Vhat,
  newdata = NULL,
  nq = 31
)
```

Arguments

`Y, X`	original training data (as passed to ‘EstBifactorTwoStep()‘); only used (and only needed) when ‘newdata=NULL‘
`clu`	original training cluster membership; only used when ‘newdata=NULL‘
`model`	same model spec as ‘Sim2Cluster()‘/‘EstBifactorTwoStep()‘
`coef`	the fitted ‘coef‘ list (‘$margins‘, ‘$copulas‘)
`Vhat`	data.frame with ‘cluster‘/‘V‘ columns, as returned by ‘condMeansBifactor()‘/‘condMediansBifactor()‘ – looked up by cluster label for each row
`newdata`	NULL (default) to re-predict the training data, or a list with ‘x‘ (new X covariates, matrix/vector) and ‘clu‘ (which fitted cluster's V to use for each new row – must already appear in ‘Vhat$cluster‘)
`nq`	number of quadrature nodes, used for BOTH the Y-quantile grid and the w-integral (default 31)

Value

data.frame, one row per observation, columns ‘EY1‘,...,‘EYd‘

`condMediansBifactor` *Conditional median of V for every cluster, all Y/X components jointly*

Description

Top-level wrapper: given the original data and a fitted bi-factor model (as returned by 'EstBifactorTwoStep()'/ 'EstBifactorCopulaGAMMFast()'), reconstructs each margin's CDF from its fitted (natural-unit) parameters, then estimates every cluster's V_k.

NOTE (perf): when the batched 'cond_median_logh_multi()' C kernel is available, this calls 'invfunc()' ONCE across ALL K clusters simultaneously (see '.prepare_bifactor_batch()'), so each of its ~40 bisection steps costs a single '.C()' call spanning whichever clusters haven't converged yet – instead of the ~40*K small '.C()' calls the original per-cluster loop made. Falls back automatically to that original per-cluster loop (via 'condMedianV_bifactor()', still using the single-cluster kernel if available) when the multi-cluster kernel can't be used.

Usage

```
condMediansBifactor(
  Y,
  X = NULL,
  clu,
  model,
  coef,
  nq = 31,
  use_c = TRUE,
  kernel_path = "bifactor_kernel.c"
)
```

Arguments

`Y, X`	N x d / N x p matrices (or vectors) of response/covariate observations; X may be NULL
`clu`	cluster membership, length N
`model`	same model spec as 'Sim2Cluster()'/'EstBifactorTwoStep()'
`coef`	the fitted 'coef' list ('$margins', '$copulas') – see the ASSUMPTIONS note at the top of this file re: natural units
`nq`	number of quadrature nodes (default 31)
`use_c`	logical (default TRUE), tries the compiled kernel(s) (batched multi-cluster first, then single-cluster), falling back to pure R
`kernel_path`	path to 'bifactor_kernel.c'/'.so'

Value

data.frame with one row per cluster: 'cluster', 'V' (conditional posterior median)

`condMedianV_bifactor` *Conditional (posterior) median of the cluster-level factor V, for ONE cluster, all margins jointly*

Description

Implements (2.9)-(2.11) for a single cluster: for each candidate v, computes every member's own per-observation density h(y_i,x_i,v) by integrating out that member's own nuisance w (the inner loop, over ALL dp margins jointly), multiplies across the n_k members (the outer product in 2.10), then finds the v at which the resulting posterior CDF equals 0.5 – using the same quadrature-rescaling + 'invfunc()'-bisection trick as 'MAP.continuous()'/'MAP.discrete()'.

Usage

```
condMedianV_bifactor(
  F1,
  F2 = NULL,
  cont,
  fam1,
  rot1,
  cpar1,
  dfC1 = NULL,
  fam2,
  rot2,
  cpar2,
  dfC2 = NULL,
  nq = 31,
  gl = NULL,
  use_c = TRUE,
  kernel_path = "bifactor_kernel.c"
)
```

Arguments

`F1`	n_k x dp matrix: F_j(x_ij) (or G_j(y_ij) for Y-margins) for this cluster's members (columns = margins, Y columns first then X, matching 'model''s order)
`F2`	n_k x dp matrix: the left-limit CDF, needed only for discrete margins (columns for continuous margins can be NA)
`cont`	logical, length dp: TRUE = continuous (uses copula density), FALSE = discrete (uses copula CDF difference)
`fam1`, `rot1`, `cpar1`, `dfC1`	length-dp vectors: copula-1 (links each margin to V) family/rotation/parameter/df
`fam2`, `rot2`, `cpar2`, `dfC2`	length-dp vectors: copula-2 (links each margin's V-conditional value to the per-observation nuisance w) family/rotation/parameter/df

nq	number of Gauss-Legendre-on-(0,1) quadrature nodes, used for BOTH the v-integral and each member's own w-integral (default 31, matching MAP.continuous()/MAP.discrete())
gl	optional precomputed ‘statmod::gauss.quad.prob(nq)‘ result, to avoid recomputing it once per cluster (‘condMediansBifactor()‘ supplies this)
use_c	logical (default TRUE). If TRUE, tries to use the compiled ‘cond_median_logh()‘ C kernel (see ‘kernel_path‘) for the expensive nested member/margin/w-node computation, falling back to the pure-R version automatically if the kernel can't be found/compiled/loaded. Set to FALSE to force the pure-R path (e.g. for debugging).
kernel_path	path to ‘bifactor_kernel.c‘ (or its already-compiled ‘.so‘/‘.dll‘) – the SAME file ‘EstBifactorCopulaGAMMFast()‘ uses, since ‘cond_median_logh()‘ was added alongside its existing ‘bifactor_negloglik()‘. Defaults to ‘"bifactor_kernel.c"‘ in the working directory.

Value

scalar: the conditional posterior median of V_k for this cluster

---

`cpar2link` *Link to copula parameter*

Description

Computes the copula parameters given a linear combination of covariates.

Usage

```
cpar2link(cpar, family = "clayton")
```

Arguments

cpar	copula parameters
family	copula family: "normal (gaussian)" , "t" , "clayton" , "frank" , "gumbel", "fgm", "galambos", "huesler-reiss", "joe", "plackett", "ind" .

Details

NOTE (fix): the previous version grouped ‘"clayton"‘ together with ‘"huesler-reiss"‘/‘"galambos"‘ under ‘th = log(cpar)‘. That's the correct inverse for huesler-reiss/galambos (whose forward link in ‘linkCop()‘ is ‘cpar = 1/expf = exp(th)‘), but NOT for clayton, whose forward link is instead ‘cpar = 2*tau/(1-tau)‘ with ‘tau = 1/(1+exp(-th))‘ (the SAME tau clayton shares with gumbel/joe/ plackett, just combined into a different final formula) – inverting that gives ‘th = log(cpar/2)‘, not ‘log(cpar)‘. The old formula was systematically wrong by a factor of exactly 2 in ‘cpar‘ (equivalently ‘log(2)‘ in ‘th‘) for every clayton value: ‘linkCop(log(cpar), "clayton")$cpar‘ evaluates to ‘2*cpar‘, not ‘cpar‘. Also added the ‘"fgm"‘ case, previously missing entirely even though ‘linkCop()‘ supports it (‘cpar = tanh(th)‘, the same link as gaussian/t).

Value

th	unrestricted copula parameters

Examples

```
th = cpar2link(0.5,"gaussian")
```

---

dlap *Laplace pdf*

Description

This function computes the pdf of Laplace distribution

Usage

```
dlap(x)
```

Arguments

x	vector of responses

Value

f	Laplace cdf

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

EstBifactorCopulaGAMMFast *Bi-factor copula estimation for clustered mixed data (fast version)*

Description

This is a faster replacement for ‘EstBifactorCopulaGAMM()‘. It is NOT numerically identical to the original: the continuous-margin branch of the likelihood has been corrected to match eq. (2.8) of the bi-factor copula paper (see the note below), so expect the estimates to differ from ‘EstBifactorCopulaGAMM()‘ whenever the model includes a continuous margin. Everything else about the objective function is unchanged; the only remaining differences are *how* it is evaluated.

Why the original is slow: ‘nlm()‘ evaluates the objective function many times (it estimates the gradient numerically, so every iteration costs roughly ‘npar+1‘ evaluations of ‘likf()‘, and ‘iterlim=200‘ allows many iterations). Each call to the original ‘likf()‘ loops over every cluster in R and, for every cluster, calls ‘dcop()‘/‘pcond()‘ – which each re-dispatch through a ‘switch()‘ on the family

name – on freshly allocated vectors. That per-call R-interpreter overhead, multiplied by (number of clusters) x (number of 'nlm' evaluations), is what dominates the runtime; the actual arithmetic is tiny.

This version:

1. Precomputes, once (outside the objective function), everything that does not change across 'nlm' evaluations: the cluster grouping (row order + boundaries), and the integer family/ rotation/df codes for every copula slot.
2. Moves the per-cluster double-Gauss-quadrature loop (previously re-run, in R, on every objective evaluation) into a single call to a compiled C routine, 'bifactor_negloglik()', defined in 'bifactor_kernel.c'. That C routine reproduces the arithmetic of 'dcop()'/'pcond()' exactly (including the 'pcond.R' "fgm" rotation case), with one intentional correctness fix: the continuous-margin branch of the original 'likf()' computed 'temp1 = matff[,j]*dcop(u1,vv,...)' – the density prefactor 'f_j(x_j)*c_[j,1][F_j(x_j),v]' required by eq. (2.8) of the bi-factor copula paper – but then discarded it and multiplied by the conditional cdf 'temp2' in its place. Both this C kernel and the pure-R fallback below now correctly use 'temp1'/'f_j*c_[j,1]' as the multiplier instead.
3. If the C library cannot be found/compiled/loaded, this function automatically falls back to a pure-R evaluator (still with the precomputed cluster grouping, which is a safe, free win on its own) and prints a one-time message explaining how to compile the C kernel for the full speedup.
4. Parallelizes the C kernel's cluster loop with OpenMP (clusters are statistically independent, so this is embarrassingly parallel and exact – not an approximation). This is the single biggest remaining lever if estimation is still slow: with 'method="nlminb"', the numerical gradient costs '2*length(start)' evaluations of the C kernel per outer iteration, so on a machine with several cores this can give a near-linear additional speedup on top of the C rewrite itself. Compiled automatically by '.load_bifactor_kernel()'; falls back to a single-threaded build if the compiler doesn't support '-fopenmp'.

Usage

```
EstBifactorCopulaGAMMFast(
  Y,
  X = NULL,
  clu,
  model,
  start = NULL,
  LB = NULL,
  UB = NULL,
  nq = 25,
  prediction = TRUE,
  kernel_path = "bifactor_kernel.c",
  method = c("nlminb", "nlm"),
  compute_se = TRUE
)
```

Arguments

Y	matrix of responses (assumed continuous or discrete).

X	matrix of covariates (assumed continuous or discrete).
clu	variable of size n defining the clusters; can be a factor
model	list of lists for columns vectors of x. For any j, model[[j]]$margin is a list containing the margin family, param (vector of parameters), and size if necessary.
start	list of vector of starting point for the estimation.
LB	list of vector of lower bound for the parameters.
UB	list of vector of upper bound for the parameters.
nq	number of nodes and weights for Gaussian quadrature of the product of conditional copulas; default is 25.
prediction	logical variable for prediction of latent variables V (default is TRUE).
kernel_path	Path to the compiled shared library (‘.so‘/‘.dll‘) or to ‘bifactor_kernel.c‘ (in which case it will be compiled with ‘R CMD SHLIB‘ the first time). Defaults to ‘"bifactor_kernel.c"‘ in the current working directory.
method	Optimizer to use. ‘"nlminb"‘ (default) enforces ‘LB‘/‘UB‘ natively and uses a bound-aware numerical gradient – recommended, and the fix for the "numerical gradients are very large / doesn’t converge" failure mode that can occur with ‘"nlm"‘ (kept only for comparison / backward compatibility), which is unconstrained and can have its finite-difference gradient corrupted whenever a probe step crosses a tight bound and hits the internal ‘1e100‘ infeasibility penalty.
compute_se	logical (default TRUE). If TRUE, also computes ‘se‘/‘tval‘/‘pval‘ via a numerical Hessian – an extra O(npar^2) evaluations of ‘likf()‘ on top of the estimation itself. Set to FALSE to skip this and get only point estimates back faster (‘se‘/‘tval‘/‘pval‘ will be ‘NULL‘).

Value

list with ‘coef‘ (list with ‘copulas‘/‘margins‘ matrices, same shape as ‘EstBifactorTwoStep()‘’s ‘coef‘ – ‘margins‘ is in natural units via ‘link2parM()‘, matching that function’s own convention, not the raw pre-link scale Step 1 optimizes over) and ‘gradient‘ (shaped like ‘coef‘: a list with ‘copula‘ and ‘margin‘ matrices, each ‘dp x 2‘ with the same row order as ‘coef$copulas‘/ ‘coef$margins‘, rather than a flat vector. Same parameter scale as ‘EstBifactorTwoStep()‘’s ‘gradient‘ – Step 1’s copula-1 gradient is chain-rule-converted from the raw copula-parameter scale it’s actually computed on), plus ‘se‘/‘tval‘/‘pval‘ (same ‘list(copula=,margin=)‘ shape as ‘coef‘, standard errors and asymptotic Wald tests – see ‘se_note‘ for whether these are the naive two-step version or the full-MLE version), ‘se_note‘ (a short string explaining which), ‘loglik‘, ‘aic‘, ‘bic‘. These all come from ‘final‘ (the refined joint fit) when ‘refine=TRUE‘; otherwise they’re assembled directly from the Step 1 + Step 2 results – in that case ‘se‘/‘tval‘/‘pval‘ are the NAIVE two-step version (Step 1’s margin/cpar1 standard errors reused from each ‘EstMargin1Factor()‘ fit in isolation; Step 2’s cpar2 standard error treats margin/cpar1 as fixed/known), which understate the true uncertainty – use ‘refine=TRUE‘ for valid inference. Also returns ‘gradient_copulas‘, a ‘dp x 2‘ matrix (same row order as ‘coef$copulas‘) with columns ‘"step1"‘ (the chain-rule-converted cpar1 gradient from ‘EstMargin1Factor()‘) and ‘"step2"‘ (the cpar2 gradient from the Step 2 fit) – always the two-step breakdown, regardless of ‘refine‘, since that distinction doesn’t exist once/if the joint refinement runs. Also returns ‘step1‘ (per-margin ‘EstMargin1Factor()‘ results), ‘step2‘ (the copula-2-only fit), ‘warm_start‘ (the assembled full parameter vector – also used to warm-start the refinement step when ‘refine=TRUE‘), ‘final‘ (the refined joint fit if ‘refine=TRUE‘, else ‘NULL‘), and ‘calc_time‘ (a named vector with the wall-clock time for each phase).

`EstBifactorTwoStep` *Two-step ("IFM"-style) estimation for the bi-factor copula GAMM*

Description

Implements the two-step estimator suggested by section 2.1 of the bi-factor copula paper: "the parameters of Gj and Dj,1 can be estimated using the CRAN package CopulaGAMM ... with no covariates," applied separately to every margin (Step 1, via ‘EstMargin1Factor()‘, with an analytic gradient built from ‘margins()‘/‘coplik()‘), followed by plugging those fixed values into the full bi-factor likelihood to estimate only the copula-2 parameters (Step 2, reusing the existing C kernel, ‘bifactor_negloglik()‘, but now optimizing over just ‘dp‘ parameters instead of the whole model).

This changes the *shape* of the optimization, not just its constant: Step 1 replaces one big, badly-scaled joint optimization with ‘dp‘ small, cheap, single-quadrature ones (no integral over the second latent factor W at all); Step 2 keeps the double quadrature but only has to differentiate ‘dp‘ parameters instead of the full parameter count. Both are typically much cheaper than one joint fit over everything at once.

IMPORTANT CAVEAT: two-step (IFM-style) estimators are consistent but not fully efficient – standard errors read off Step 2 alone understate the true uncertainty, because they treat the Step 1 estimates as known constants rather than propagating their sampling variability. If you need valid inference (not just point estimates), either use a sandwich/Godambe covariance, bootstrap over clusters, or (recommended, and the default here) use these estimates purely as a warm start for one final joint optimization over all parameters (‘refine=TRUE‘): since it starts essentially at the optimum, that final pass typically converges in a handful of iterations rather than a full cold-start search, so you get IFM-level speed for the hard part and exact joint- MLE point estimates/standard errors at the end.

Usage

```
EstBifactorTwoStep(
  Y,
  X = NULL,
  clu,
  model,
  start = NULL,
  LB = NULL,
  UB = NULL,
  nq = 25,
  prediction = TRUE,
  kernel_path = "bifactor_kernel.c",
  refine = TRUE,
  compute_se = TRUE
)
```

Arguments

`Y`	matrix of responses (assumed continuous or discrete).

X	matrix of covariates (assumed continuous or discrete).
clu	variable of size n defining the clusters; can be a factor
model	list of lists for columns vectors of x. For any j, model[[j]]$margin is a list containing the margin family, param (vector of parameters), and size if necessary.
start	list of vector of starting point for the estimation.
LB	list of vector of lower bound for the parameters.
UB	list of vector of upper bound for the parameters.
nq	number of nodes and weights for Gaussian quadrature of the product of conditional copulas; default is 25.
prediction	logical variable for prediction of latent variables V (default is TRUE).
kernel_path	Path to the compiled shared library (‘.so‘/‘.dll‘) or to ‘bifactor_kernel.c‘ (in which case it will be compiled with ‘R CMD SHLIB‘ the first time). Defaults to ‘"bifactor_kernel.c"‘ in the current working directory.
refine	logical (default TRUE). If TRUE, runs one final joint ‘EstBifactorCopulaGAMMFast()‘ optimization over *all* parameters, warm-started from the Step 1 + Step 2 estimates. If FALSE, returns the two-step estimates directly (faster, but see the efficiency caveat above).
compute_se	logical (default TRUE). If TRUE, also computes ‘se‘/‘tval‘/‘pval‘ (passed through to ‘EstMargin1Factor()‘ and, if ‘refine=TRUE‘, to ‘EstBifactorCopulaGAMMFast()‘ too, plus this function’s own Hessian for cpar2 when ‘refine=FALSE‘) – several extra O(n^2)-ish Hessian evaluations on top of the estimation itself. Set to FALSE to skip all of that and get only point estimates back faster (‘se‘/‘tval‘/‘pval‘ will be ‘NULL‘).

Value

list with ‘coef‘ (list with ‘copulas‘/‘margins‘ matrices, same shape as ‘EstBifactorCopulaGAMMFast()‘’s ‘coef‘ – ‘margins‘ is in natural units via ‘link2parM()‘, matching that function’s own convention, not the raw pre-link scale Step 1 optimizes over) and ‘gradient‘ (shaped like ‘coef‘: a list with ‘copula‘ and ‘margin‘ matrices, each ‘dp x 2‘ with the same row order as ‘coef$copulas‘/‘coef$margins‘, rather than a flat vector. Same parameter scale as ‘EstBifactorCopulaGAMMFast()‘’s ‘gradient‘ – Step 1’s copula-1 gradient is chain-rule-converted from the raw copula-parameter scale it’s actually computed on), plus ‘se‘/‘tval‘/‘pval‘ (same ‘list(copula=,margin=)‘ shape as ‘coef‘, standard errors and asymptotic Wald tests – see ‘se_note‘ for whether these are the naive two-step version or the full-MLE version), ‘se_note‘ (a short string explaining which), ‘loglik‘, ‘aic‘, ‘bic‘. These all come from ‘final‘ (the refined joint fit) when ‘refine=TRUE‘; otherwise they’re assembled directly from the Step 1 + Step 2 results – in that case ‘se‘/‘tval‘/‘pval‘ are the NAIVE two-step version (Step 1’s margin/cpar1 standard errors reused from each ‘EstMargin1Factor()‘ fit in isolation; Step 2’s cpar2 standard error treats margin/cpar1 as fixed/known), which understate the true uncertainty – use ‘refine=TRUE‘ for valid inference. Also returns ‘gradient_copulas‘, a ‘dp x 2‘ matrix (same row order as ‘coef$copulas‘) with columns ‘"step1"‘ (the chain-rule-converted cpar1 gradient from ‘EstMargin1Factor()‘) and ‘"step2"‘ (the cpar2 gradient from the Step 2 fit) – always the two-step breakdown, regardless of ‘refine‘, since that distinction doesn’t exist once/if the joint refinement runs. Also returns ‘step1‘ (per-margin ‘EstMargin1Factor()‘ results), ‘step2‘ (the copula-2-only fit), ‘warm_start‘ (the assembled full parameter vector – also used to warm-start the refinement step when ‘refine=TRUE‘), ‘final‘ (the refined joint fit if ‘refine=TRUE‘, else ‘NULL‘), and ‘calc_time‘ (a named vector with the wall-clock time for each phase).

`EstMargin1Factor` *Step 1 of the two-step bi-factor estimator: single-margin, 1-factor fit*

Description

Estimates, for one response or covariate margin j in isolation, its marginal-distribution parameters and the parameter of the copula linking it to the cluster-level latent factor V, using only the univariate 1-factor clustered-data likelihood of eq. (2.6)/(2.7) of the bi-factor copula paper – i.e. every other margin and the second-level latent factor W are ignored entirely. This is exactly Step 1 of the two-step estimator described in that section: "the parameters of Gj and Dj,1 can be estimated using the CRAN package CopulaGAMM ... with no covariates," applied separately to each margin.

Compared to plugging this margin into the full bi-factor joint likelihood, this is cheap in two independent ways: (1) it needs only a *single* Gauss-Legendre quadrature over V (no integral over W at all), and (2) it only has ‘nth+1‘ parameters (the margin’s own 1-2 parameters plus one copula parameter) instead of the full model’s parameter count, so even a finite-difference gradient over just this piece is cheap.

Where possible, the gradient used by ‘nlm()‘ is *analytic*, built from the closed-form derivatives already implemented in ‘margins()‘ (margin cdf/pdf and their parameter derivatives) and ‘coplik()‘ (copula conditional-cdf/density and their derivatives), combined via the chain rule – the same style ‘mlecop()‘ in ‘coplik.R‘ already uses for the simple i.i.d. case, extended here to the clustered/latent-factor likelihood (a quadrature-weighted mixture over V).

As with ‘EstBifactorCopulaGAMMFast()‘, the per-cluster/quadrature reduction (previously an R-level loop) is offloaded to a compiled C kernel, ‘margin1factor_negloglik_grad()‘ in ‘margin1factor_kernel.c‘, whenever ‘cop1‘ is one of the families ‘coplik()‘ covers. That kernel also parallelizes across clusters with OpenMP (clusters are independent here too), the same way the joint model’s kernel does. ‘margins()‘ itself still runs in R (a single vectorized call per evaluation, not a per-cluster loop, so porting every margin family to C would cost more than it saves); only the reduction and the copula-side derivatives are in C. Falls back automatically to the pure-R evaluator below if the kernel can’t be found/compiled/loaded.

Coverage note: ‘coplik()‘ provides analytic derivatives for copula families "gaussian"/"normal", "t", "clayton", "frank", "gumbel", "joe", "plackett", "fgm". It does not cover "galambos", "hueslerreiss", or "ind". For a continuous margin with one of the covered families, the gradient below is fully analytic (both the margin-parameter and the copula-parameter pieces). For a discrete margin, ‘coplik()‘ does not expose an analytic d/du of the conditional cdf C(u|v) (needed for the margin-parameter piece via the chain rule through u=G(y;theta)), so that piece falls back to a cheap central difference – still only 1-2 extra copula evaluations, since this is a 2-3 parameter problem, not a finite-difference gradient over the whole model. For a copula family not covered by ‘coplik()‘ at all, the whole gradient falls back to ‘nlm()‘’s own finite-difference gradient (via ‘dcop()‘/‘pcond()‘), which is still cheap here for the same reason.

Usage

```
EstMargin1Factor(
  z,
  clu,
```

```
  margin,
  cop1,
  rot1 = 0,
  dfM = 0,
  dfC1 = 0,
  nq = 25,
  start,
  LB,
  UB,
  kernel_path = "margin1factor_kernel.c",
  compute_se = TRUE
)
```

Arguments

z	the N observations for this single margin (one column of Y or X).
clu	cluster membership, length N (same convention as 'EstBifactorCopulaGAMM').
margin	margin family string, as used by 'margins()'/'margins0()'.
cop1	copula-1 family string, as used by 'coplik()'/'dcop()'/'pcond()'.
rot1	rotation for copula 1 (0/90/180/270, default 0).
dfM	Student margin degrees of freedom (0 if not applicable).
dfC1	Student copula-1 degrees of freedom (0 if not applicable).
nq	number of Gauss-Legendre-on-(0,1) quadrature nodes (default 25).
start	starting values, 'c(margin params..., cpar1)'.
LB, UB	bounds, same length/order as 'start'.
kernel_path	Path to the compiled Step-1 C kernel ('margin1factor_kernel.c', or its already-compiled '.so'/'.dll'). Defaults to '"margin1factor_kernel.c"' in the current working directory; compiled automatically (with an OpenMP attempt, same as 'EstBifactorCopulaGAMMFast()''s kernel) the first time it's needed. Set to 'NULL' to force the pure-R evaluator (e.g. for debugging).
compute_se	logical (default TRUE). If TRUE, also computes 'se'/'tval'/'pval' via a numerical Hessian – an extra O(n^2) evaluations of the objective on top of the estimation itself. Set to FALSE to skip this and get only point estimates back faster ('se'/'tval'/'pval' will be 'NULL').

Value

list with 'param' ('c(margin params, cpar1)' at the optimum, raw/pre-link2parM scale), 'coef' ('margin' converted to natural units via 'link2parM()', and 'cpar1' – already natural units, no conversion needed since this function optimizes cpar1 directly, not the joint model's linked "param" scale), 'se'/'tval'/'pval' (same shape as 'coef', standard errors from a numerical Hessian of the negative log-likelihood with the margin part delta-method-converted to match coef's natural units; 'tval' is an asymptotic Wald statistic, not a finite-sample Student-t, so 'pval' uses the normal approximation), 'loglik', 'gradient' (shaped like 'coef': a list with 'margin' and 'cpar1', on the RAW optimization scale – a convergence diagnostic, not meant to pair with 'coef''s units the way 'se'/'tval'/'pval'

do), ‘mle‘ (the raw ‘nlm()‘ object), and ‘used_c_kernel‘ (whether the compiled kernel was actually used). NOTE: the ‘se‘/‘tval‘/‘pval‘ fallback path (for objectives without an analytic gradient attribute) calls ‘.central_grad()‘, defined in ‘EstBifactorCopulaGAMMFast.R‘ – source that file too if calling ‘EstMargin1Factor()‘ on its own.

`expcpdf` *Exponential cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
expcpdf(z, th)
```

Arguments

`z`	vector of responses
`th`	th is mean > 0

Value

`out`	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

`geomcpdf` *Geometric with p = 1/(1+exp(-th)) cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
geomcpdf(z, th)
```

Arguments

`z`	vector of responses
`th`	linear combination of covariates (can be negative)

Value

out	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

---

invfuncNew *Inverse function*

Description

This function is used to get the inverse of a monotonic function on (0,1), depending on parameters, and using the bisection method

Usage

```
invfuncNew(q, func, th, lb = 1e-12, ub = 1 - 1e-12, tol = 1e-08, nbreak = 40)
```

Arguments

q	Function value (can be a vector if func() supports a vector argument)
func	Function of one argument to be inverted
th	Function parameters
lb	Lower bound for the possible values
ub	Upper bound for the possible values
tol	Tolerance for the inversion
nbreak	Maximum number of iterations (default is 40)

Details

NOTE: renamed from ‘invfunc()‘ to ‘invfuncNew()‘. This package Imports ‘CopulaGAMM‘, which also exports a function named ‘invfunc()‘ – and unlike ‘pcond()‘/‘linkCop()‘ (which turned out to be identical to CopulaGAMM’s own and were dropped in favor of importing them directly), THIS function contains a real, necessary bug fix CopulaGAMM’s release doesn’t have (see the NOTE (fix) comment below: the three ‘func0()‘ calls used to pass the internal ‘iconv‘ convergence-tracking vector in place of ‘th‘, which is what caused the "object ’hder’ not found" / node-stack-overflow bugs earlier in this project’s history). Every caller in this package (‘qcondNew()‘’s family-specific helpers, ‘condMedianV_bifactor()‘, ‘condMediansBifactor()‘, ‘EstBifactorTwoStep()‘) has been updated to call ‘invfuncNew()‘ explicitly, so none of them can silently fall back to CopulaGAMM’s still-buggy original.

Value

out	Inverse values

Author(s)

Pavel Krupskii

`lapcpdf` *Laplace cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
lapcpdf(z, th)
```

Arguments

`z`	vector of responses
`th`	th[,1] is mean, th[,2] is standard deviation > 0

Value

`out`	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

`link2parM` *Parameter values from link parameter for margins*

Description

Computation of parameter values for margins from the associated link parameters

Usage

```
link2parM(th, marg)
```

Arguments

`th`	Link vector
`marg`	Margin family: "binomial" (bernoulli), "poisson", "exponential", "normal".

Value

parM Associated margin parameters

Examples

```
p=link2parM(.43,"bernoulli")
```

`margins0` *Margins cdf/pdf*

Description

This function computes the cdf and pdf

Usage

```
margins0(z, th, model, dfM)
```

Arguments

`z`	vector of responses
`th`	2-d link (can be negative)
`model`	model for margin: "binomial" (bernoulli), "poisson", "nbinom" (mean is the parameter),"nbinom1" (p is the parameter), "geometric", "multinomial", "exponential", "weibull", "normal","t", "laplace"
`dfM`	degrees of freedom for the Student margin (default is 0)

Value

`out`	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

Examples

```
out = margins0(c(1:10),c(2.5,0),"poisson")
```

nbinom1cpdf *Negative binomial cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
nbinom1cpdf(z, th)
```

Arguments

z	vector of responses
th	th[,1] is size > 0 and th[,2] is mean > 0; size does not have to be integer

Value

out	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

nbinomcpdf *Negative binomial cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
nbinomcpdf(z, th)
```

Arguments

z	vector of responses
th	th[,1] is size > 0 and th[,2] is p, with 0<p<1; size does not have to be integer

Value

out	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

normcpdf *normal cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
normcpdf(z, th)
```

Arguments

z	vector of responses
th	th[,1] is mean, th[,2] is standard deviation > 0;

Value

out	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

param.dim *Number of parameters*

Description

Gives the number of paramters for a given idstribution

Usage

```
param.dim(family)
```

Arguments

family	model for margin: "binomial" (bernoulli), "poisson", "nbinom" (mean is the parameter),"nbinom1" (p is the parameter), "geometric", "multinomial", "exponential", "weibull", "normal","t", "laplace"

Value

dd	Number of parameters

Author(s)

Pavel Krupskii and Bruno N. Remillard, March 9, 2026

Examples

```
dd = param.dim("normal")
```

---

plap *Laplace cdf*

Description

This function computes the cdf of Laplace distribution

Usage

```
plap(x)
```

Arguments

x	vector of responses

Value

f	Laplace cdf

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

---

poiscpdf *Poisson cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
poiscpdf(z, th)
```

Arguments

z	vector of responses
th	values of lambda >0

Value

out	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

---

qcondcla *Inverse clayton*

Description

Inverse clayton

Usage

```
qcondcla(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter

Value

out	Conditional quantile

---

qcondfgm *Inverse FGM (B10)*

Description

Inverse FGM (B10)

Usage

```
qcondfgm(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter -1<=th<=1

Value

out	Conditional quantile

qcondfra *Inverse Frank*

Description

Inverse Frank

Usage

```
qcondfra(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter

Value

out	Conditional quantile

qcondgal *Inverse Galambos*

Description

Inverse Galambos

Usage

```
qcondgal(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter >0

Value

out	Conditional quantile

qcondgum *Inverse Gumbel*

Description

Inverse Gumbel

Usage

```
qcondgum(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter

Value

out	Conditional quantile

qcondhr *Inverse Huesler-Reiss*

Description

Inverse Huesler-Reiss

Usage

```
qcondhr(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter >0

Value

out	Conditional quantile

qcondjoe *Inverse Joe*

Description

Inverse Joe

Usage

```
qcondjoe(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter >-1

Value

out	Conditional quantile

qcondNew *Inverse conditional cdf*

Description

This function computes the quantile of conditional cdf C(U|v) for a copula C

Usage

```
qcondNew(w, v, family, cpar, rot = 0)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
family	"gaussian" , "t" , "clayton" , "fgm", "frank" , "gumbel", "plackett", "galambos", "huesler-reiss"
cpar	copula parameter (vector)
rot	rotation: 0 (default), 90, 180 (survival), or 270

Details

NOTE: renamed from ‘qcond()‘ to ‘qcondNew()‘. This package Imports ‘CopulaGAMM‘, which also exports a function named ‘qcond()‘ – keeping the same name here would create a real namespace collision (not just a load-order ambiguity, the way it was when these were loose sourced scripts), and this version fixes real bugs the ‘CopulaGAMM‘ release doesn’t have (see ‘qcondgum()‘/‘qcondjoe()‘/‘qcondgal()‘/‘qcondhr()‘’s own NOTE (fix) comments below), so silently falling back to ‘CopulaGAMM::qcond()‘ anywhere this package forgets to qualify the call would reintroduce those bugs. Every caller in this package (e.g. ‘Sim2Cluster()‘) has been updated to call ‘qcondNew()‘ explicitly.

Value

U	Conditional quantile

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

Examples

```
U = qcondNew(0.1,0.2,"gaussian",0.87)
```

---

qcondnor *Inverse Gaussian*

Description

Inverse Gaussian

Usage

```
qcondnor(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter (correlation)

Value

out	Conditional quantile

qcondpla *Inverse Plackett*

Description

Inverse Plackett

Usage

```
qcondpla(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter

Value

out	Conditional quantile

qcondt *Inverse Student*

Description

Inverse Student

Usage

```
qcondt(w, v, th)
```

Arguments

w	probability
v	value of the conditioning variable in (0,1)
th	copula parameter

Value

out	Conditional quantile

qlap *Laplace quantile*

Description

This function computes the qunatiles of Laplace distribution

Usage

```
qlap(u)
```

Arguments

u	vector of probabilities

Value

q	Laplace quantiles

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

qmargin *Quantiles of margins*

Description

This function computes the quantiles of a given margin family

Usage

```
qmargin(U, family, param, size = NULL, shape = NULL, dfM = NULL)
```

Arguments

U	vector of probabilities
family	model for margin: "binomial" (bernoulli), "poisson", "nbinom" (mean is the parameter),"nbinom1" (p is the parameter), "geometric", "multinomial", "exponential", "weibull", "normal","t", "laplace"
param	parameters
size	standard deviation if parameter of the distribution
shape	only for Weibull distribution
dfM	degrees of freedom for the Student margin (default is NULL)

Value

out	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

Examples

```
out = qmargin(0.5,"normal", param=c(10,2))
```

Sim2Cluster *Simulation of clustered data using bi-factor copulas*

Description

Generate a random sample of observations from a bi-factor copula mixed regression model.

Usage

```
Sim2Cluster(model, nclu)
```

Arguments

model	list of lists for vector Y and vector X; model$Y is a list of lists of models for the copulas and the margins and model$X is a list of list of models for the covariates; model$Y[[j]]$margin or model$X[[j]]$margin is a list containing the margin family (as provided in copulaGAMM), param (vector of parameters), and size if necessary, while model$Y[[j]]$copula[[1]] and model$Y[[j]]$copula[[2]] are lists defining the copulas (included in 'copulaGAMM' package), including the copula family name (family), the parameters (cpar), and the rotations (rot) with values in 0 (default),90,180, or 270.
nclu	vector of sample sized for clusters

Value

data	Data frame with simulated values of Y,X V, and values of clusters

Author(s)

Bruno N. Remillard

Examples

```
nclu=rep(5,25) # 25 clusters of size 5
ccY=list(list(family="gumbel",param=2),list(family="gumbel",param=1.4,rot=90) )
modY1 = list(margin=list(family="bernoulli", param=c(0.6)), copula=ccY)
modY=list(modY1)
ccX1 = list(list(family="clayton",param=0.86,rot=180),list(family="clayton",param=0.2,rot=180))
ccX2 = list(list(family="gaussian",param=0.6),list(family="clayton",param=2))
modX1 = list(margin=list(family="bernoulli", param=c(0.6)),  copula=ccX1)
modX2 = list(margin=list(family="gaussian" , param=c(10,0.2)), copula=ccX2)
modX=list(modX1,modX2)
model=list(Y=modY,X=modX)
set.seed(123)
out=Sim2Cluster(model,nclu)
```

---

tcpdf *Student cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
tcpdf(z, th, df)
```

Arguments

z	vector of responses
th	th[,1] is mean, th[,2] is standard deviation > 0
df	degrees of freedom

Value

out	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

`weibcpdf` *Weibul cdf/pdf and ders*

Description

This function computes the cdf, pdf, and associated derivatives

Usage

```
weibcpdf(z, th)
```

Arguments

`z`	vector of responses
`th`	th[,1] is rate>0, th[,2] is shape > 0;

Value

`out`	Matrix of conditional cdf, derivative with respect to parameter, pdf,

Author(s)

Pavel Krupskii and Bruno N. Remillard, January 20, 2022

Index

bounds.cop, 2
bounds.mar, 3

condMeansBifactor, 4
condMeanV_bifactor, 5
condMeanY, 6
condMeanY_given_X, 7
condMeanY_given_XV, 8
condMediansBifactor, 9
condMedianV_bifactor, 10
cpar2link, 11

dlap, 12

EstBifactorCopulaGAMMFast, 12
EstBifactorTwoStep, 15
EstMargin1Factor, 17
expcpdf, 19

geomcpdf, 19

invfuncNew, 20

lapcpdf, 21
link2parM, 21

margins0, 22

nbinom1cpdf, 23
nbinomcpdf, 23
normcpdf, 24

param.dim, 24
plap, 25
poiscpdf, 25

qcondcla, 26
qcondfgm, 26
qcondfra, 27
qcondgal, 27
qcondgum, 28
qcondhr, 28
qcondjoe, 29
qcondNew, 29
qcondnor, 30
qcondpla, 31
qcondt, 31
qlap, 32
qmargin, 32

Sim2Cluster, 33

tcpdf, 34

weibcpdf, 35