Package ‘dsfa’

July 19, 2023

Title  Distributional Stochastic Frontier Analysis
Version  2.0.2
Description  Framework to fit distributional stochastic frontier models. Casts the stochastic frontier model into the flexible framework of distributional regression or otherwise known as General Additive Models of Location, Scale and Shape (GAMLSS). Allows for linear, non-linear, random and spatial effects on all the parameters of the distribution of the output, e.g. effects on the production or cost function, heterogeneity of the noise and inefficiency. Available distributions are the normal-halfnormal and normal-exponential distribution. Estimation via the fast and reliable routines of the 'mgcv' package. For more details see <doi:10.1016/j.csda.2023.107796>.

Imports  mgcv, stats, Rdpack, Rcpp, RcppArmadillo, copula, gratia
Suggests  plm, sfaR
Depends  R (>= 3.5.0)
NeedsCompilation  yes
SystemRequirements  C++17
License  MIT + file LICENSE
LinkingTo  Rcpp, RcppArmadillo
Encoding  UTF-8
RdMacros  Rdpack
LazyData  true
RoxygenNote  7.2.3
Author  Rouven Schmidt [aut, cre]
Maintainer  Rouven Schmidt <rouven.schmidt@tu-clausthal.de>
Repository  CRAN
Date/Publication  2023-07-19 21:40:16 UTC

R topics documented:

   BurkinaFarms .............................................................. 2
   BurkinaFarms_polys ....................................................... 4
Description

Dataset of the Center for Environmental Economics and Policy in Africa (CEEPA), University of Pretoria. It contains information on the production of millet and sorghum in Burkina Faso.

Usage

BurkinaFarms
Format

BurkinaFarms is a data frame with 560 rows and 20 columns:

- **hhcode**  Respondent Households’ identification number
- **farntype**  Type of Farm Entity:
  - ‘1’ = small-scale
  - ‘2’ = medium-scale
  - ‘3’ = large-scale
- **incfarm**  Income of farmers
- **educ1**  education of household head in number of years
- **adm1**  Province/Region
- **material**  Light and heavy machinery as well as farm animal power multiplied with their price
- **qharv_millet**  Quantity of millet harvested in kg
- **lost_millet**  Quantity of millet lost due to disease and pests in kg
- **seed_millet**  Amount of seeds used for the production of millet in kg
- **land_millet**  Amount of land used for the production of millet in ha
- **labour_millet**  Total estimated number of workdays for the production of millet. One day corresponds to 6-8 hours of work completed by one individual. Household labor and hired labor are accumulated
- **fert_millet**  Amount of fertilizer used for the production of millet in kg
- **pest_millet**  Amount of pesticides used for the production of millet in kg
- **qharv_sorghum**  Quantity of sorghum harvested in kg
- **lost_sorghum**  Quantity of sorghum lost due to disease and pests in kg
- **seed_sorghum**  Amount of seeds used for the production of sorghum in kg
- **land_sorghum**  Amount of land used for the production of sorghum in ha
- **labour_sorghum**  Total estimated number of workdays for the production of sorghum. One day corresponds to 6-8 hours of work completed by one individual. Household labor and hired labor are accumulated
- **fert_sorghum**  Amount of fertilizer used for the production of sorghum in kg
- **pest_sorghum**  Amount of pesticides used for the production of sorghum in kg

Details

This is a subset of the data which contains data only from Burkina Faso for the outputs sorghum and millet.

Source

<https://figshare.com/collections/An_agricultural_survey_for_more_than_9_500_African_households/1574094>

References

**Description**

Dataset of Burkina Faso - Subnational Administrative Boundaries

**Usage**

BurkinaFarms_polys

**Format**

This file contains the polygons of the regions of Burkina Faso.

**Source**

<https://data.humdata.org/dataset/burkina-faso-administrative-boundaries>

---

**Description**

Inverse cumulative distribution function

**Usage**

cdf2quantile(p, cdf, interval = c(-3, 3), ...)

**Arguments**

- `p` numeric vector of probabilities.
- `cdf` function, cumulative distribution function which to invert.
- `interval` numeric vector of length 2, determining the lower and upper bound of the unit-root interval
- `...` other arguments for the cdf, e.g. mu, sigma_v, sigma_u, s...

**Details**

Code is a clone from the package gbutils.

**Value**

Numeric vector of p evaluated in the inverse cdf.
Examples

```r
q=5
cdf <- pnorm(q=q, mean=1, sd=2)
q_numeric <- cdf2quantile(p=cdf, cdf=pnorm, mean=1, sd=2)
all.equal(q, q_numeric)
```

---

### chainrule

**Description**

Chainrule for derivs objects.

**Usage**

```r
chainrule(f_list, tri, deriv_order)
```

**Arguments**

- `f_list`: list of derivs objects of length `M`, e.g. `list(f_1(·), f_2(·), ..., f_M(·))`
- `tri`: list; created by the function `trind_generator()`.
- `deriv_order`: integer; maximum order of derivative. Available are 0, 2 and 4.

**Details**

Let \( f_m \) be a function defined in \([\text{trind()}\]) where \( m \in 1, ..., M \). Define \( h((x_{n1}, x_{n2}, ..., x_{nK})) = f_1(·) \circ f_2(·) \circ ... \circ f_M(x_{n1}, x_{n2}, ..., x_{nK}) \). In order to get the derivatives of \( h(·) \) w.r.t all parameters \( x_{nk} \), the chainrule is applied. For more details see \([\text{trind()}\]) and \([\text{trind_generator()}\])

**Value**

Returns an object of class derivs for the function \( h(·) \).

**See Also**

Other derivs: `derivs_transform()`, `differencerule()`, `ind2joint()`, `list2derivs()`, `productrule()`, `quotientrule()`, `sumrule()`, `trind_generator()`, `trind()`

**Examples**

```r
A<-matrix(c(1:9)/10, ncol=1)
A_derivs<-list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=4)
B_derivs<-transform(A, type="exp", par=0, deriv_order=4)
C_derivs<-transform(B_derivs, type="log", par=0, deriv_order=4)
chainrule(list(C_derivs, B_derivs), trind_generator(1), deriv_order=4) #equal to A_derivs
```
comper

Description

The comper implements the composed-error distribution in which the $\mu$, $\sigma_V$ and $\sigma_U$ can depend on additive predictors. Useable with mgcv::gam, the additive predictors are specified via a list of formulae.

Usage

comper(
  link = list("identity", "logshift", "logshift"),
  s = -1,
  distr = "normhnorm",
  b = 0.01
)

Arguments

- **link**: three item list, specifying the link for the $\mu$, $\sigma_V$ and $\sigma_U$ parameters. See details.
- **s**: integer; $s = -1$ for production and $s = 1$ for cost function.
- **distr**: string; determines the distribution:
  - 'normhnorm', Normal-halfnormal distribution
  - 'normexp', Normal-exponential distribution
- **b**: positive parameter of the logshift link function.

Details

Used with gam() to fit distributional stochastic frontier model. The function is called with a list containing three formulae:

1. The first formula specifies the response on the left hand side and the structure of the additive predictor for $\mu$ parameter on the right hand side. Link function is "identity".
2. The second formula is one sided, specifying the additive predictor for the $\sigma_V$ on the right hand side. Link function is "logshift", e.g. $\log\{\sigma_V\} + b$.
3. The third formula is one sided, specifying the additive predictor for the $\sigma_U$ on the right hand side. Link function is "logshift", e.g. $\log\{\sigma_U\} + b$.

The fitted values and linear predictors for this family will be three column matrices. The first column is the $\mu$, the second column is the $\sigma_V$, the third column is $\sigma_U$. For more details of the distribution see dcomper().

Value

An object inheriting from class general.family of the mgcv package, which can be used in the mgcv and dsfa package.
References


Examples

```r
### First example with simulated data
# Set seed, sample size and type of function
set.seed(1337)
N=500 # Sample size
s=-1 # Set to production function

# Generate covariates
x1<-runif(N,-1,1); x2<-runif(N,-1,1); x3<-runif(N,-1,1)
x4<-runif(N,-1,1); x5<-runif(N,-1,1)

# Set parameters of the distribution
mu=2+0.75*x1+0.4*x2+0.6*x2^2+6*log(x3+2)^(1/4) # production function parameter
sigma_v=exp(-1.5+0.75*x4) # noise parameter
sigma_u=exp(-1+sin(2*pi*x5)) # inefficiency parameter

# Simulate responses and create dataset
y<-rcomper(n=N, mu=mu, sigma_v=sigma_v, sigma_u=sigma_u, s=s, distr="normhnorm")
dat<-data.frame(y, x1, x2, x3, x4, x5)

# Write formulae for parameters
mu_formula<-y~x1+x2+I(x2^2)+s(x3, bs="ps")
sigma_v_formula<~1+x4
sigma_u_formula<~1+s(x5, bs="ps")

# Fit model
model<-dsfa(formula=list(mu_formula, sigma_v_formula, sigma_u_formula),
              data=dat, family=comper(s=s, distr="normhnorm"), optimizer = c("efs"))

# Model summary
summary(model)
```
# Smooth effects
# Effect of x3 on the predictor of the production function
plot(model, select=1)  # Estimated function
lines(x3[order(x3)], 6*log(x3[order(x3)]+2)^(1/4)-
mean(6*log(x3[order(x3)]+2)^(1/4)), col=2)  # True effect

# Effect of x5 on the predictor of the inefficiency
plot(model, select=2)  # Estimated function
lines(x5[order(x5)], -1+sin(2*pi*x5)[order(x5)]-
mean(-1+sin(2*pi*x5)), col=2)  # True effect

### Second example with real data

data("RiceFarms", package = "plm")  # load data
RiceFarms[,c("goutput","size","seed", "totlabor", "urea")]<-
log(RiceFarms[,c("goutput","size","seed", "totlabor", "urea")])  # log outputs and inputs
RiceFarms$id<-factor(RiceFarms$id)  # id as factor

# Set to production function
s=-1

# Write formulae for parameters
mu_formula<-'goutput ~ s(size, bs="ps") + s(seed, bs="ps") + # non-linear effects
s(totlabor, bs="ps") + s(urea, bs="ps") + # non-linear effects
varieties + # factor
s(id, bs="re")  # random effect
sigma_v_formula<-'1
sigma_u_formula<-'bimas

# Fit model with normhnorm distribution
model<-'dsfa(formula=list(mu_formula, sigma_v_formula, sigma_u_formula),
data=RiceFarms, family=comper(s=-1, distr="normhnorm"), optimizer = "efs")

# Summary of model
summary(model)

# Plot smooths
plot(model)

### Third example with real data of cost function

data("electricity", package = "sfaR")  # load data

# Log inputs and outputs as in Greene 1990 eq. 46
electricity$lcof<-log(electricity$cost/electricity$fprice)
electricity$lo<-log(electricity$output)
electricity$llf<-log(electricity$lprice/electricity$fprice)
electricity$lcf<-log(electricity$cprice/electricity$fprice)

# Set to cost function
s=1
#Write formulae for parameters
mu_formula<-lcof ~ s(lo, bs="ps") + s(1lf, bs="ps") + s(1cf, bs="ps") #non-linear effects
sigma_v_formula<-~1
sigma_u_formula<-~s(lo, bs="ps") + s(1share, bs="ps") + s(cshare, bs="ps")

#Fit model with normhnorm distribution
model<-dsfa(formula=list(mu_formula, sigma_v_formula, sigma_u_formula),
data=electricity, family=comper(s=s, distr="normhnorm"),
optimizer = "efs")

#Summary of model
summary(model)

#Plot smooths
plot(model)

---

comper_mv
comper

### Description

The comper implements the multivariate composed-error distribution in which the \( \mu_1, \sigma_{V1}, \sigma_{U2}, \mu_2, \sigma_{V2}, \sigma_{U2} \) and \( \delta \) can depend on additive predictors. Useable with mgcv::gam, the additive predictors are specified via a list of formulae.

### Usage

```r
comper_mv(
  link = list("identity", "logshift", "logshift", "identity", "logshift", "logshift",
               "glogit"),
  s = c(-1, -1),
  distr = c("normhnorm", "normhnorm", "normal"),
  rot = 0,
  b = 0.01
)
```

### Arguments

- **link** seven item list, specifying the links for \( \mu_1, \sigma_{V1}, \sigma_{U2}, \mu_2, \sigma_{V2}, \sigma_{U2} \) and \( \delta \). See details.
- **s** integer vector of length two; each element corresponds to one marginal.
- **distr** string vector of length three; the first two elements determine the distribution of the marginals. Available are:
  - ‘normhnorm’, Normal-halfnormal distribution
  - ‘normexp’, Normal-exponential distribution
  - The last element determines the distribution of the copula:
`'independent'`, Independence copula
`'normal'`, Gaussian copula
`'clayton'`, Clayton copula
`'gumbel'`, Gumbel copula
`'frank'`, Frank copula
`'joe'`, Joe copula
`'amh'`, Ali-Mikhail-Haq copula

```
rot integer determining the rotation for Archimedian copulas. Can be 90, 180 or 270.
b positive parameter of the logshift link function.
```

Details

Used with `gam()` to fit distributional stochastic frontier model. The function is called with a list containing three formulae:

1. The first formula specifies the response of marginal one on the left hand side and the structure of the additive predictor for $\mu_1$ parameter on the right hand side. Link function is "identity".
2. The second formula is one sided, specifying the additive predictor for the $\sigma_{V1}$ on the right hand side. Link function is "logshift", e.g. $\log(\sigma_{V1}) + b$.
3. The third formula is one sided, specifying the additive predictor for the $\sigma_{U1}$ on the right hand side. Link function is "logshift", e.g. $\log(\sigma_{U1}) + b$.
4. The fourth formula specifies the response of marginal two on the left hand side and the structure of the additive predictor for $\mu_2$ parameter on the right hand side. Link function is "identity".
5. The fifth formula is one sided, specifying the additive predictor for the $\sigma_{V2}$ on the right hand side. Link function is "logshift", e.g. $\log(\sigma_{V2}) + b$.
6. The sixth formula is one sided, specifying the additive predictor for the $\sigma_{U2}$ on the right hand side. Link function is "logshift", e.g. $\log(\sigma_{U2}) + b$.
7. The seventh formula is one sided, specifying the additive predictor for the $\delta$ on the right hand side. Link function is "glogit".

The fitted values and linear predictors for this family will be seven column matrices. For more details of the distribution see `dcomper()`.

Value

An object inheriting from class `general.family` of the mgcv package, which can be used in the `mgcv` and `dsfa` package.

References


**Examples**

```r
# Set seed, sample size and type of function
set.seed(1337)
N=1000  # Sample size
s<-c(-1,-1)  # Set to production function for margin 1 and set to cost function for margin 2

distr_cop="normal"
distr_marg1="normhnorm"
distr_marg2="normhnorm"

# Generate covariates
x1<-runif(N,-1,1); x2<-runif(N,-1,1); x3<-runif(N,-1,1)
x4<-runif(N,-1,1); x5<-runif(N,-1,1); x6<-runif(N,-1,1)
x7<-runif(N,-1,1)

mu1=6+2*x1+(-2/3)*x1^2  # production function parameter 1
sigma_v1=exp(-1.5+sin(2*pi*x2))  # noise parameter 1
sigma_u1=exp(-1)  # inefficiency parameter 1

mu2=5*x4^2+4*log(x4+2)^(1/4)  # cost function parameter 2
sigma_v2=exp(-1.5)  # noise parameter 2
sigma_u2=exp(-1+sin(2*pi*x6))  # inefficiency parameter 2

delta=transform(x=matrix(1+2.5*cos(4*x7)),
                type="glogitinv",
                par=delta_bounds(distr_cop), deriv_order = 0)

# Simulate responses and create dataset
Y<-rcomper_mv(n=N, mu=cbind(mu1,mu2),
              sigma_v=cbind(sigma_v1, sigma_v2),
              sigma_u = cbind(sigma_u1, sigma_u2), s=s,
              delta=delta,
              distr = c(distr_marg1,distr_marg2,distr_cop))
dat<-data.frame(y1=Y[,1], y2=Y[,2], x1, x2, x3, x4, x5, x6, x7)

# Write formulae for parameters
mu_1_formula<-y1~s(x1,bs="ps")
sigma_v1_formula<~s(x2,bs="ps")
sigma_u1_formula<~1
mu_2_formula<-y2~s(x4,bs="ps")
sigma_v2_formula<~1
```
comper_mv

sigma_u2_formula <- ~s(x6, bs = "ps")
delta_formula <- ~s(x7, bs = "ps")

# Fit model
model <- dsfa(formula = list(mu_1_formula, sigma_v1_formula, sigma_u1_formula, 
                mu_2_formula, sigma_v2_formula, sigma_u2_formula, 
                delta_formula), data = dat, 
                family = comper_mv(s = s, distr = c(distr_marg1, distr_marg2, distr_cop)), 
                optimizer = "efs")

# Model summary
summary(model)

# Smooth effects
# Effect of x1 on the predictor of the production function of margin 1
plot(model, select = 1) # Estimated function
lines(x1[order(x1)], 2*x1[order(x1)] + (-1/3)*x1[order(x1)]^2 - 
      mean(2*x1 + (-1/3)*x1^2), col = 2) # True effect

# Effect of x2 on the predictor of the noise of margin 1
plot(model, select = 2) # Estimated function
lines(x2[order(x2)], -1.5 + sin(2*pi*x2[order(x2)]) - 
      mean(-1.5 + sin(2*pi*x2)), col = 2) # True effect

# Effect of x4 on the predictor of the production function of margin 2
plot(model, select = 3) # Estimated function
lines(x4[order(x4)], 3 + 5*x4[order(x4)]^2 + 4*log(x4[order(x4)]+2)^(1/4) - 
      mean(3 + 5*x4^2 + 4*log(x4+2)^(1/4)), col = 2) # True effect

# Effect of x6 on the predictor of the inefficiency of margin 2
plot(model, select = 4) # Estimated function
lines(x6[order(x6)], -1 + sin(2*pi*x6[order(x6)]) - 
      mean(-1 + sin(2*pi*x6)), col = 2) # True effect

# Effect of x7 on the predictor of the copula
plot(model, select = 5) # Estimated function
lines(x7[order(x7)], 2.5*cos(4*x7[order(x7)]) - 
      mean(2.5*cos(4*x7)), col = 2) # True effect

efficiency(model)
elasticity(model)

# Second example with real data

data(BurkinaFarms)
data(BurkinaFarms_polys)

# Write formulae for parameters
mu_1_formula <- qharv_millet ~ s(land_millet, bs = "ps") + s(labour_millet, bs = "ps") + 
                 s(material, bs = "ps") + s(fert_millet, bs = "ps") + 
                 s(adm1, bs = "mrf", xt = BurkinaFarms_polys)

sigma_v1_formula <- ~1

sigma_u1_formula <- farmtype + s(pest_millet, bs = "ps")
mu_2_formula<-qharv_sorghum~s(land_sorghum, bs="ps")+s(labour_sorghum, bs="ps")+s(material, bs="ps")+s(fert_sorghum, bs="ps")+s(adm1, bs="mrf", xt=BurkinaFarms_polys)
sigma_v2_formula<-1
sigma_u2_formula<-~farmtype+s(pest_sorghum, bs="ps")

delta_formula<-~1

model<-dsfa(formula=list(mu_1_formula, sigma_v1_formula, sigma_u1_formula, mu_2_formula, sigma_v2_formula, sigma_u2_formula, delta_formula), data=BurkinaFarms, family=comper_mv(s=c(-1,-1), distr=c("normhnorm","normhnorm","normal")), optimizer="efs")

plot(model)

Description
The cop implements multiple copula distributions in which the parameter $\delta$ can depend on additive predictors. Useable with mgcv::gam, the additive predictors are specified via a formula.

Usage

cop(link = list("glogit"), W, distr = "normal", rot = 0)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>formula, specifying the link for $\delta$ parameter. See details.</td>
</tr>
<tr>
<td>W</td>
<td>numeric matrix of pseudo observations. Must have two columns.</td>
</tr>
<tr>
<td>distr</td>
<td>string, defines the copula family: <code>independent</code> = Independence copula, <code>normal</code> = Gaussian copula, <code>clayton</code> = Clayton copula, <code>gumbel</code> = Gumbel copula, <code>frank</code> = Frank copula, <code>joe</code> = Joe copula, <code>amh</code> = Ali-Mikhail-Haq copula</td>
</tr>
<tr>
<td>rot</td>
<td>integer determining the rotation for Archimedian copulas. Can be 90, 180 or 270.</td>
</tr>
</tbody>
</table>
Details

Mostly internal function. Used with gam to fit copula model, which in turn is used for starting values. The function gam is from the mgcv package and is called with a formula. The formula specifies a dummy on the left hand side and the structure of the additive predictor for the $\delta$ parameter on the right hand side. Link function is "generalized logit", where for each distr argument there are specific min and max arguments, which are the boundaries of the parameter space. Although the parameter space is larger in theory for some copulas, numeric under- and overflow limits the parameter space. The intervals for the parameter delta are provided by [delta_bounds()]. WARNING: Only the estimates of the coefficients are useful. The rest of the 'mgcv' object has no meaningful values, as gam() was more or less abused here.

Value

An object inheriting from class general.family of the mgcv package, which can be used in the 'mgcv' and 'dsfa' package.

References


See Also

Other copula: dcop(), delta_bounds()

Examples

```r
#Set seed, sample size and type of copula
set.seed(1337)
N=500 #Sample size
cop="gumbel" #copula
rot=180 #rotation

#Generate covariates
x1<-runif(N,-1,1); x2<-runif(N,-1,1)
```
# Set parameters of the copula
eta <- matrix(1 + 2.5 * x1 + 1.75 * sin(pi * x2), nrow = N)
delta <- transform(x = eta, type = "glogitinv", par = as.numeric(delta_bounds(cop)), deriv_order = 0)

# Simulate pseudo observations W and create dataset
dat <- as.data.frame(rcop(n = N, delta = delta, distr = cop, rot = rot))
dat$y <- 1 # Add dummy response variable

# Write formulae for parameters
delta_formula <- y ~ x1 + s(x2, bs = "ps")

# Fit model
model <- mgcv::gam(delta_formula, data = dat,
                   family = cop(W = dat[, 1:2],
                                distr = cop, rot = rot),
                   optimizer = "efs")

# Smooth effects
# Effect of x2 on the predictor of delta
plot(model, select = 1) # Estimated function
lines(x2[order(x2)], 1.75 * sin(pi * x2[order(x2)]) - mean(1.75 * sin(pi * x2)), col = 2) # True effect

---

dcomper

Composed-Error distribution

**Description**

Probability density function, distribution, quantile function and random number generation for the composed-error distribution

**Usage**

```r
dcomper(
  x,
  mu = 0,
  sigma_v = 1,
  sigma_u = 1,
  s = -1,
  distr = "normhnorm",
  deriv_order = 0,
  tri = NULL,
  log.p = FALSE
)
pcomper(
```
q,  
mu = 0,  
sigma_v = 1,  
sigma_u = 1,  
s = -1,  
distr = "normhnorm",  
deriv_order = 0,  
tri = NULL,  
log.p = FALSE
)

qcomper(  
p,  
mu = 0,  
sigma_v = 1,  
sigma_u = 1,  
s = -1,  
distr = "normhnorm",  
log.p = FALSE
)

rcomper(n, mu = 0, sigma_v = 1, sigma_u = 1, s = -1, distr = "normhnorm")

Arguments

x numeric vector of quantiles.  
mu numeric vector of \( \mu \).  
sigma_v numeric vector of \( \sigma_V \). Must be positive.  
sigma_u numeric vector of \( \sigma_U \). Must be positive.  
s integer; \( s = -1 \) for production and \( s = 1 \) for cost function.  
distr string; determines the distribution:  
'normhnorm', Normal-halfnormal distribution  
'normexp', Normal-exponential distribution

deriv_order integer; maximum order of derivative. Available are 0,2 and 4.  
tri optional; index matrix for upper triangular, generated by trind_generator.  
log.p logical; if TRUE, probabilities p are given as log(p).  
q numeric vector of quantiles.  
p numeric vector of probabilities.  
n positive integer; number of observations.

Details

This is wrapper function for the normal-halfnormal and normal-exponential distribution. A random variable \( X \) follows a composed error distribution if \( X = V + s \cdot U \), where \( V \sim N(\mu, \sigma_V^2) \) and \( U \sim HN(0, \sigma_U^2) \) or \( U \sim Exp(\sigma_U^2) \). For more details see dnormhnorm and dnormexp. Here, \( s = -1 \) for production and \( s = 1 \) for cost function.
Value
dcomper() gives the density, pcomper() gives the distribution function, qcomper() gives the quantile function, and rcomper() generates random numbers, with given parameters. dcomper() and pcomper() returns a derivs object.

Functions
- pcomper(): distribution function for the composed-error distribution.
- qcomper(): quantile function for the composed-error distribution.
- rcomper(): random number generation for the composed-error distribution.

References

See Also
Other distribution: dcomper_mv(), dnormexp(), dnormhnorm()

Examples
```r
df <- dcomper(x=5, mu=1, sigma_v=2, sigma_u=3, s=-1, distr="normhnorm")
cdf <- pcomper(q=5, mu=1, sigma_v=2, sigma_u=3, s=-1, distr="normhnorm")
q <- qcomper(p=seq(0.1, 0.9, by=0.1), mu=1, sigma_v=2, sigma_u=3, s=-1, distr="normhnorm")
r <- rcomper(n=10, mu=1, sigma_v=2, sigma_u=3, s=-1, distr="normhnorm")
```

**dcomper_mv**

Multivariate Composed-Error distribution

Description
Probablity density function, distribution, quantile function and random number generation for the multivariate composed-error distribution
Usage

dcomper_mv(
  x,
  mu = matrix(c(0, 0), ncol = 2),
  sigma_v = matrix(c(1, 1), ncol = 2),
  sigma_u = matrix(c(1, 1), ncol = 2),
  delta = matrix(0, nrow = 1),
  s = c(-1, -1),
  distr = c("normhnorm", "normhnorm", "normal"),
  rot = 0,
  deriv_order = 0,
  tri = NULL,
  log.p = FALSE
)

pcomper_mv(
  q,
  mu = matrix(c(0, 0), ncol = 2),
  sigma_v = matrix(c(1, 1), ncol = 2),
  sigma_u = matrix(c(1, 1), ncol = 2),
  delta = 0,
  s = c(-1, -1),
  distr = c("normhnorm", "normhnorm", "normal"),
  rot = 0,
  deriv_order = 0,
  tri = NULL,
  log.p = FALSE
)

rcomper_mv(
  n,
  mu = matrix(c(0, 0), ncol = 2),
  sigma_v = matrix(c(1, 1), ncol = 2),
  sigma_u = matrix(c(1, 1), ncol = 2),
  delta = matrix(0, nrow = 1),
  s = c(-1, -1),
  distr = c("normhnorm", "normhnorm", "normal"),
  rot = 0
)

Arguments

x    numeric matrix of quantiles. Must have two columns.
mu   numeric matrix of μ. Must have two columns.
sigma_v numeric matrix of σ_V. Must be positive and have two columns.
sigma_u numeric matrix of σ_U. Must be positive and have two columns.
delta numeric vector of copula parameter δ.
s

distr

rot

deriv_order

tri

log.p

q

n

Details

A bivariate random vector \((X_1, X_2) = X\) follows a multivariate composed-error distribution \(f_{X_1, X_2}(x_1, x_2)\), which can be rewritten using Sklar's theorem via a copula

\[
f_{X_1, X_2}(y_1, y_2) = c(F_{X_1}(x_1), F_{X_2}(x_2), \delta) \cdot f_{X_1}(x_1)f_{X_2}(x_2),
\]

where \(c(\cdot)\) is the density of the copula and \(F_{X_m}(x_m), f_{X_m}(x_m)\) are the marginal cdfs and pdfs respectively for \(m \in \{1, 2\}\). \(\delta\) is the copula parameter.

Value

dcomper_mv gives the density, pcomper_mv give the distribution function, and rcomper_mv generates random numbers, with given parameters. If the derivatives are calculated the output is a derivs object.

Functions

- pcomper_mv(): distribution function for the multivariate composed-error distribution.
- rcomper_mv(): random number generation for the multivariate composed-error distribution.

References

dcop

Copula function

Description

Probablitiy density function, distribution and random number generation for copulas.

Usage

dcop(
  W,
  delta,
  distr = "normal",
  rot = 0,
  deriv_order = 0,
  tri = NULL,
  log.p = FALSE
)
dcop

```r
pcop(W, delta = 0, distr = "normal", rot = 0, log.p = FALSE)
rcop(n, delta = 0, distr = "normal", rot = 0)
```

**Arguments**

- `W` numeric matrix of pseudo observations. Must have two columns.
- `delta` numeric vector of copula parameter $\delta$.
- `distr` string, defines the copula family:
  - `"independent"` = Independence copula
  - `"normal"` = Gaussian copula
  - `"clayton"` = Clayton copula
  - `"gumbel"` = Gumbel copula
  - `"frank"` = Frank copula
  - `"joe"` = Joe copula
  - `"amh"` = Ali-Mikhail-Haq copula
- `rot`, integer determining the rotation for Archimedean copulas. Can be 90, 180 or 270.
- `deriv_order` integer; maximum order of derivative. Available are 0,2 and 4.
- `tri` optional; index matrix for upper triangular, generated by `trind_generator`.
- `log.p` logical; if TRUE, probabilities p are given as log(p).
- `n` positive integer; number of observations.

**Details**

A two-dimensional copula $C(w_1, w_2, \delta)$ is a multivariate cumulative distribution function for which the marginal probability distribution of $w_1$ and $w_2$ are uniform on the interval $[0, 1]$. The parameter $\delta$ specifies the copula.

The functions `pcop()` and `rcop()` are wrapper functions for the `pCopula` and `rCopula`.

**Value**

- `dcop` gives the density,
- `pcop` gives the distribution function for a specified copula and
- `rcop` generates random numbers, with given `delta`. `dcop()` returns a `derivs` object. For more details see `trind` and `trind_generator`.

**Functions**

- `pcop()`: distribution function for copula.
- `rcop()`: random number generation for copula.
References


See Also

Other copula: `cop()`, `delta_bounds()`

Examples

```r
u=0.3; v=0.7; p=0.5
pdf <- dcop(W=cbind(u,v), delta=p, distr="normal")
cdf <- pcop(W=cbind(u,v), delta=p, distr="normal")
r <- rcop(n=100, delta=p, distr="normal")
```

---

**delta_bounds**

**Bounds of Copula Parameter delta**

**Description**

Provides the minimum and maximum of the parameter space for $\delta$

**Usage**

`delta_bounds(distr)`

**Arguments**

- `distr` string, defines the copula family:
  - `independent` = Independence copula
  - `normal` = Gaussian copula
  - `clayton` = Clayton copula
  - `gumbel` = Gumbel copula
  - `frank` = Frank copula
  - `joe` = Joe copula
  - `amh` = Ali-Mikhail-Haq copula

**Details**

Although the parameter space is larger in theory for some copulas, numeric under- and overflow limits the parameter space. The parameter space of $\delta$ is specified for each copula below:

- ‘independent’, min=0 and max=1
• 'normal', min=-1 and max=1
• 'clayton', min=1e-16 and max=28
• 'gumbel', min=1 and max=17
• 'frank', min=-35 and max=35
• 'joe', min=1e-16 and max=30
• 'amh', min=-1 and max=1

Value

Returns numeric vector of length two with first argument being the minimum and the second argument being the maximum of the parameter space.

See Also

Other copula: cop(), dcop()

Examples

delta_bounds("normal")

dervs_transform

derivs_transform

Description

Transforms a derivs object via the specified function and applies the chainrule.

Usage

derivs_transform(f, type, par, tri, deriv_order)

Arguments

f derivs object.

(type string, specifies the transformation function. Available are:
1. 'identity': \( f(x) = x \).
2. 'exp': \( f(x) = \exp\{x\} \).
3. 'log': \( f(x) = \log\{x\} \).
4. 'glogit': \( f(x) = \log\{(-x + \text{min})/(\text{max} - x)\} \), where \( \text{par} = (\text{min}, \text{max}) \).
5. 'glogitinv': \( f(x) = \exp\{x\} \cdot (\text{max} + \text{min})/(1 + \exp\{x\}) \), where \( \text{par} = (\text{min}, \text{max}) \).
6. 'inv': \( f(x) = \frac{1}{x} \).
7. 'pnorm': \( f(x) = \Phi(x) \).
8. 'qnorm': \( f(x) = \Phi^{-1}(x) \).
9. ‘mexp’: \( f(x) = -\exp\{x\} \).
10. ‘zeta’: \( f(x) = \log\{2 \cdot \Phi(x)\} \).
11. ‘constant’: \( f(x) = c \).
12. ‘chainrule_utility’: \( f(x) = f'(x) = f''(x) = f'''(x) = f''''(x) \).
13. onemx: \( 1 - x \)

\( \text{par} \) numeric vector, additional parameters, e.g. min and max for \( \text{glogit} \).
\( \text{tri} \) list; created by the function [\text{trind_generator()}].
\( \text{deriv_order} \) integer; maximum order of derivative. Available are 0, 2 and 4.

Details

Takes the \( \text{derivs} \) object \( f \) as an input for the function specified by \( \text{type} \) and evaluates it together with the derivatives utilizing the chainrule. For more details see [\text{trind()}] and [\text{trind_generator()}].

Value

Returns an object of class \( \text{derivs} \)

See Also

Other \( \text{derivs} \): \text{chainrule()}, \text{differencerule()}, \text{ind2joint()}, \text{list2derivs()}, \text{productrule()}, \text{quotientrule()}, \text{sumrule()}, \text{trind_generator()}, \text{trind()}

Examples

\begin{verbatim}
A<-matrix(c(1:9)/10, ncol=1)
A_mat<-list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=4)
derivs_transform(f = derivs_transform(f = A, type="exp", par=0,
                                 tri=trind_generator(1), deriv_order=4),
                 type="log", par=0, tri=trind_generator(1), deriv_order=4)
\end{verbatim}

---

differencerule

Description

Differencerule for derivs objects.

Usage

differencerule(f_list, tri, deriv_order)

Arguments

\( f\text{\_list} \) list of \( \text{derivs} \) objects of length \( M \), e.g. \( \text{list}(f_1(\cdot), f_2(\cdot), ..., f_M(\cdot)) \)
\( \text{tri} \) list; created by the function [\text{trind_generator()}].
\( \text{deriv\_order} \) integer; maximum order of derivative. Available are 0, 2 and 4.
Details

Let \( f_m \) be a function defined in [trind()], where \( m \in 1, ..., M \). Define \( h((x_{n1}, x_{n2}, ..., x_{nK})) = f_1(\cdot) - f_2(\cdot) ... - f_M(x_{n1}, x_{n2}, ..., x_{nK})) \). In order to get the derivatives of \( h(\cdot) \) w.r.t all parameters \( x_{nk} \), the difference rule is applied. For more details see [trind()] and [trind_generator()].

Value

Returns an object of class derivs for the function \( h(\cdot) \).

See Also

Other derivs: chainrule(), derivs_transform(), ind2joint(), list2derivs(), productrule(), quotientrule(), sumrule(), trind_generator(), trind()

Examples

\[
A<-matrix(c(1:9)/10, ncol=1)
A_derivs<-list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=4)
differencerule(list(A_derivs, A_derivs), trind_generator(1), deriv_order=4) \#equal to 0
\]

\[\]

\( dnormexp \)  
Normal-Exponential distribution

Description

Probabilty density function, distribution, quantile function and random number generation for the normal-exponential distribution.

Usage

\[
dnormexp(x, 
  mu = 0, 
  sigma_v = 1, 
  sigma_u = 1, 
  s = -1, 
  deriv_order = 0, 
  tri = NULL, 
  log.p = FALSE
)
\]

\[
pnormexp(q, 
  mu = 0, 
  sigma_v = 1, 
  sigma_u = 1, 
)
\]
\begin{verbatim}
s = -1, 
deriv_order = 0, 
tri = NULL, 
log.p = FALSE
)
qnormexp(p, mu = 0, sigma_v = 1, sigma_u = 1, s = -1, log.p = FALSE)
rnormexp(n, mu = 0, sigma_v = 1, sigma_u = 1, s = -1)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{x} \hfill numeric vector of quantiles.
  \item \texttt{mu} \hfill numeric vector of \( \mu \).
  \item \texttt{sigma_v} \hfill numeric vector of \( \sigma_V \). Must be positive.
  \item \texttt{sigma_u} \hfill numeric vector of \( \sigma_U \). Must be positive.
  \item \texttt{s} \hfill integer; \( s = -1 \) for production and \( s = 1 \) for cost function.
  \item \texttt{deriv_order} \hfill integer; maximum order of derivative. Available are 0, 2 and 4.
  \item \texttt{tri} \hfill optional; index matrix for upper triangular, generated by \texttt{trind_generator}.
  \item \texttt{log.p} \hfill logical; if TRUE, probabilities \texttt{p} are given as log(p).
  \item \texttt{q} \hfill numeric vector of quantiles.
  \item \texttt{p} \hfill numeric vector of probabilities.
  \item \texttt{n} \hfill positive integer; number of observations.
\end{itemize}

Details

A random variable \( X \) follows a normal-exponential distribution if \( X = V + s \cdot U \), where \( V \sim N(\mu, \sigma_V^2) \) and \( U \sim Exp(\sigma_U) \). The density is given by

\[ f_X(x) = \frac{\sigma_U}{2} \exp\left\{ \sigma_U(s\mu) + \frac{1}{2}\sigma_U^2\sigma_V^2 - \sigma_U(sx) \right\} 2\Phi\left( \frac{1}{\sigma_V}(-s\mu) - \sigma_U\sigma_V + \frac{1}{\sigma_V}(sx) \right) , \]

where \( s = -1 \) for production and \( s = 1 \) for cost function. In the latter case the distribution is equivalent to the Exponentially modified Gaussian distribution.

Value

\texttt{dnormexp()} gives the density, \texttt{pnormexp()} give the distribution function, \texttt{qnormexp()} gives the quantile function, and \texttt{rnormexp()} generates random numbers, with given parameters. \texttt{dnormexp()} and \texttt{pnormexp()} return a \texttt{derivs} object. For more details see \texttt{trind} and \texttt{trind_generator}.

Functions

- \texttt{pnormexp()}: distribution function for the normal-exponential distribution.
- \texttt{qnormexp()}: quantile function for the normal-exponential distribution.
- \texttt{rnormexp()}: random number generation for the normal-exponential distribution.
dnormhnorm

References


See Also

Other distribution: dcomper_mv(), dcomper(), dnormhnorm()

Examples

```r
pdf <- dnormexp(x=5, mu=1, sigma_v=2, sigma_u=3, s=-1)
cdf <- pnormexp(q=5, mu=1, sigma_v=2, sigma_u=3, s=-1)
q <- qnormexp(p=seq(0.1, 0.9, by=0.1), mu=1, sigma_v=2, sigma_u=3, s=-1)
r <- rnormexp(n=10, mu=1, sigma_v=2, sigma_u=3, s=-1)
```

---

**dnormhnorm**

*Normal-halfnormal distribution*

**Description**

Probability density function, distribution, quantile function and random number generation for the normal-halfnormal distribution

**Usage**

```r
dnormhnorm(
  x,  # X
  mu = 0,  # Location parameter
  sigma_v = 1,  # Scale parameter for the normal component
  sigma_u = 1,  # Scale parameter for the halfnormal component
  s = -1,  # Shape parameter
  deriv_order = 0,  # Derivative order
  tri = NULL,  # Trimming
  log.p = FALSE  # Logarithm of probability
)
```

```r
pnormhnorm(
```

```r
```
\[ f_X(x) = \frac{1}{\sqrt{\sigma_V^2 + \sigma_U^2}} \phi\left( \frac{x - \mu}{\sqrt{\sigma_V^2 + \sigma_U^2}} \right) \Phi\left( s \frac{\sigma_U}{\sigma_V} \frac{x - \mu}{\sqrt{\sigma_V^2 + \sigma_U^2}} \right), \]

where \( s = -1 \) for production and \( s = 1 \) for cost function.

### Value

dnormhnorm() gives the density, pnormhnorm() give the distribution function, qnormhnorm() gives the quantile function, and rnormhnorm() generates random numbers, with given parameters. dnormhnorm() and pnormhnorm() return a derivs object. For more details see trind and trind_generator.
Functions

- `pnormhnorm()`: distribution function for the normal-halfnormal distribution.
- `qnormhnorm()`: quantile function for the normal-halfnormal distribution.
- `rnormhnorm()`: random number generation for the normal-halfnormal distribution.

References


See Also

Other distribution: `dcomper_mv()`, `dcomper()`, `dnormexp()`

Examples

```r
pdf <- dnormhnorm(x=5, mu=1, sigma_v=2, sigma_u=3, s=-1)
cdf <- pnormhnorm(q=5, mu=1, sigma_v=2, sigma_u=3, s=-1)
q <- qnormhnorm(p=seq(0.1, 0.9, by=0.1), mu=1, sigma_v=2, sigma_u=3, s=-1)
r <- rnormhnorm(n=10, mu=1, sigma_v=2, sigma_u=3, s=-1)
```

dsfa

*dsfa-package: Distributional Stochastic Frontier Analysis*

Description

The dsfa package implements the specification, estimation and prediction of distributional stochastic frontier models via mgcv. The basic distributional stochastic frontier model is given by:

\[ Y_n = \eta^\mu(x_n^\mu) + V_n + s \cdot U_n \]

where \( n \in \{1, 2, ..., N\} \). \( V_n \) and \( U_n \) are the noise and (in)efficiency respectively.

- For \( s = -1 \), \( \eta^\mu(\cdot) \) is the production function and \( x_n^\mu \) are the log inputs. Alternatively, if \( s = 1 \), \( \eta^\mu(\cdot) \) is the cost function and \( x_n^\mu \) are the log cost. The vector \( x_n^\mu \) may also contain other variables.
• The noise is represented as \( V_n \sim N(0, \sigma_{\nu \theta}^2) \), where \( \sigma_{\nu \theta} = \exp(\eta_{\nu \theta}^\nu(x_{\nu \theta}^\nu)) \). Here, \( x_{\nu \theta}^\nu \) are the observed covariates which influence the parameter of the noise.

• The (in)efficiency can be represented in two ways.
  - If \( U_n \sim HN(\sigma_{U \theta}^2) \), where \( \sigma_{U \theta} = \exp(\eta_{U \theta}^\nu(x_{U \theta}^\nu)) \). Here, \( x_{U \theta}^\nu \) are the observed covariates which influence the parameter of the (in)efficiency. Consequently:
    \[
    Y_n \sim \text{normhnorm}(\mu_n = \eta^\mu(x_n^\mu), \sigma_{V \theta} = \exp(\eta_{V \theta}^\nu(x_{V \theta}^\nu)), \sigma_{U \theta} = \exp(\eta_{U \theta}^\nu(x_{U \theta}^\nu)), s = s)
    \]
    For more details see \texttt{dnormhnorm}.
  - If \( U_n \sim \text{Exp}(\sigma_{U \theta}) \), where \( \sigma_{U \theta} = \exp(\eta_{U \theta}^\nu(x_{U \theta}^\nu)) \). Here, \( x_{U \theta}^\nu \) are the observed covariates which influence the parameter of the (in)efficiency. Consequently:
    \[
    Y_n \sim \text{normexp}(\mu_n = \eta^\mu(x_n^\mu), \sigma_{V \theta} = \exp(\eta_{V \theta}^\nu(x_{V \theta}^\nu)), \sigma_{U \theta} = \exp(\eta_{U \theta}^\nu(x_{U \theta}^\nu)), s = s)
    \]
    For more details see \texttt{dnormexp}.

Consequently, \( Y_n \) follows a composed-error distribution. For an overview see \texttt{dcomper}.

Let \( \theta \) be a parameter of the distribution of \( Y_n \), e.g. \( \theta \in \{\mu_n, \sigma_{U \theta}, \sigma_{V \theta}\} \). Further, let \( g_{\theta}^{-1}(\cdot) \) be the monotonic response function, which links the additive predictor \( \eta(x_{\theta}^\nu) \) to the parameter space for the parameter \( \theta \) via the additive model:

\[
g_{\theta}^{-1}(\theta) = \eta(x_{\theta}^\nu) = \beta_0^\theta + \sum_{j=1}^{J^\theta} h_{j \theta}(x_{n j}^\nu)
\]

Thus, the additive predictor \( \eta(x_{\theta}^\nu) \) is made up by the intercept \( \beta_0^\theta \) and \( J^\theta \) smooths terms. The \texttt{mgcv} packages provides a framework for fitting distributional regression models. For more information see \texttt{comper}. The additive predictors can be defined via formulae in \texttt{gam}. Within the formulae for the parameter \( \theta \), the smooth predictor for the variable \( x_{n j}^\nu \) can be specified via the function \( s \), which is \( h_{j \theta}(\cdot) \) in the notation above. The smooth functions may be:

• linear effects, may include polynomials or regression splines.
• non-linear effects, which can be modeled via penalized regression splines, e.g. \texttt{p.spline, tprs}.
• random effects, \texttt{random.effects}.
• spatial effects, which can be modeled via \texttt{mrf}.

An overview is provided at \texttt{smooth.terms}. The functions \texttt{gam, predict.gam} and \texttt{plot.gam}, are alike to the basic S functions. A number of other functions such as \texttt{summary.gam, residuals.gam} and \texttt{anova.gam} are also provided, for extracting information from a fitted \texttt{gamObject}.

The main functions are:

• \texttt{comper} Object which can be used to fit a composed-error stochastic frontier model with the \texttt{mgcv} package.
• \texttt{comper_mv} Object which can be used to fit a multivariate composed-error stochastic frontier model with the \texttt{mgcv} package.
• \texttt{elasticity} Calculates and plots the elasticity of a smooth function.
• \texttt{efficiency} Calculates the expected technical (in)efficiency index \( E[U|E] \) or \( E[\exp(-U)|E] \).
Further useful functions are:

- `dcomper` Probability density function, distribution, quantile function and random number generation for the composed-error distribution.
- `dcomper_mv` Probability density function, distribution, quantile function and random number generation for the multivariate composed-error distribution.
- `dcop` Probability density function, distribution and random number generation for copulas.

These are written in C++ for fast and accurate evaluation including derivatives. They may be helpful for other researchers, who want to avoid the tedious implementation. Additionally:

- `cop` Object which can be used to fit a copula with the `mgcv` package.

**Usage**

```r
dsfa(
  formula,
  family = comper(link = list("identity", "logshift", "logshift"), s = -1, distr = "normhnorm"),
  data = list(),
  optimizer = "efs",
  ...
)
```

**Arguments**

- `formula` A list of formulas specifying the additive predictors. See `formula.gam` and `gam.models` for more details.
- `family` The family object specifies the (multivariate) composed-error distribution and link of the model. See `comper` and `comper_mv` for more details.
- `data` A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from `environment(formula)`: typically the environment from which `dsfa` is called.
- `optimizer` An array specifying the numerical optimization method to use to optimize the smoothing parameter estimation criterion (given by `method`). "outer" for the more stable direct approach. "outer" can use several alternative optimizers, specified in the second element of `optimizer`: "newton" (default), "bfgs", "optim", "nlm" and "nlm.fd" (the latter is based entirely on finite differenced derivatives and is very slow). "efs" for the extended Fellner Schall method of Wood and Fasiolo (2017).
- `...` other parameters of `gam`

**Details**

This function is a wrapper for `gam`.

**Value**

Returns a `gam` object.
## Author(s)

- Rouven Schmidt <rouven.schmidt@tu-clausthal.de>

## References


## Examples

```r
### First example with simulated data
#Set seed, sample size and type of function
set.seed(1337)
N=500 #Sample size
s=-1 #Set to production function

#Generate covariates
x1<-runif(N,-1,1); x2<-runif(N,-1,1); x3<-runif(N,-1,1)
x4<-runif(N,-1,1); x5<-runif(N,-1,1)

#Set parameters of the distribution
mu=2+0.75*x1+0.4*x2+0.6*x2^2+6*log(x3+2)^(1/4) #production function parameter
sigma_v=exp(-1.5+0.75*x4) #noise parameter
sigma_u=exp(-1+sin(2*pi*x5)) #inefficiency parameter

#Simulate responses and create dataset
y<-rcomper(n=N, mu=mu, sigma_v=sigma_v, sigma_u=sigma_u, s=s, distr="normhnorm")
dat<-data.frame(y, x1, x2, x3, x4, x5)

#Write formulae for parameters
mu_formula<-y~x1+x2+I(x2^2)+s(x3, bs="ps")
sigma_v_formula<-~1+x4
sigma_u_formula<-~1+s(x5, bs="ps")

#Fit model
model<-dsfa(formula=list(mu_formula, sigma_v_formula, sigma_u_formula),
data=dat, family=comper(s=s, distr="normhnorm"), optimizer = c("efs"))

#Model summary
summary(model)
```
#Smooth effects

## Effect of $x_3$ on the predictor of the production function

```r
plot(model, select=1)  # Estimated function
lines(x3[order(x3)], 6*log(x3[order(x3)]+2)^(1/4) - mean(6*log(x3[order(x3)]+2)^(1/4)), col=2)  # True effect
```

## Effect of $x_5$ on the predictor of the inefficiency

```r
plot(model, select=2)  # Estimated function
lines(x5[order(x5)], -1+sin(2*pi*x5)[order(x5)] - mean(-1+sin(2*pi*x5)), col=2)  # True effect
```

### Second example with real data of production function

```r
data("RiceFarms", package = "plm")  # load data
RiceFarms[,c("goutput","size","seed", "totlabor", "urea")]<-
  log(RiceFarms[,c("goutput","size","seed", "totlabor", "urea")])  # log outputs and inputs
RiceFarms$id<-factor(RiceFarms$id)  # id as factor

# Set to production function
s=-1

# Write formulae for parameters
mu_formula<-goutput ~ s(size, bs="ps") + s(seed, bs="ps") + # non-linear effects
  s(totlabor, bs="ps") + s(urea, bs="ps") + # non-linear effects
  varieties + # factor
  s(id, bs="re")  # random effect
sigma_v_formula<-~1
sigma_u_formula<-~bimas

# Fit model with normhmnorm distribution
model<-dsfa(formula=list(mu_formula, sigma_v_formula, sigma_u_formula),
data=RiceFarms, family=comper(s=s, distr="normhmnorm"), optimizer = "efs")

# Summary of model
summary(model)

# Plot smooths
plot(model)
```

### Third example with real data of cost function

```r
data("electricity", package = "sfaR")  # load data

# Log inputs and outputs as in Greene 1990 eq. 46
electricity$lcof<-log(electricity$cost/electricity$fprice)
electricity$lo<-log(electricity$output)
electricity$llf<-log(electricity$lprice/electricity$fprice)
electricity$lcf<-log(electricity$cprice/electricity$fprice)

# Set to cost function
s=1

# Write formulae for parameters
efficiency

```r
mu_formula<-lcof ~ s(lo, bs="ps") + s(llf, bs="ps") + s(lcf, bs="ps") # non-linear effects
sigma_v_formula<-~1
sigma_u_formula<-~s(lo, bs="ps") + s(lshare, bs="ps") + s(cshare, bs="ps")

# Fit model with normhnorm distribution
model<-dsfa(formula=list(mu_formula, sigma_v_formula, sigma_u_formula),
             data=electricity, family=comper(s=s, distr="normhnorm"), optimizer = "efs")

# Summary of model
summary(model)

# Plot smooths
plot(model)
```

---

**Description**

Calculates the expected technical (in)efficiency index.

**Usage**

```r
efficiency(object, alpha = 0.05, type = "jondrow")
```

**Arguments**

- **object**: fitted mgcv object with family `comper()` or `comper_mv()`.
- **alpha**: for the 

\[(1 - \alpha) \cdot 100\% \text{ confidence interval. Must be in } (0,1).\]

- **type**: default is "jondrow" for \(E[U|\mathcal{E}]\), alternatively "battese" for \(E[\exp(-U)|\mathcal{E}]\).

**Value**

Returns a matrix of the expected (in)efficiency estimates as well the lower and upper bound of the 

\[(1 - \alpha) \cdot 100\% \text{ confidence interval.}\]

**References**


**Examples**

```r
# Set seed, sample size and type of function
set.seed(1337)
N=500 # Sample size
s=-1 # Set to production function

# Generate covariates
x1<-runif(N,-1,1); x2<-runif(N,-1,1); x3<-runif(N,-1,1)
x4<-runif(N,-1,1); x5<-runif(N,-1,1)

# Set parameters of the distribution
mu=2+0.75*x1+0.4*x2+0.6*x2^2+6*log(x3+2)^(1/4) # production function parameter
sigma_v=exp(-1.5+0.75*x4) # noise parameter
sigma_u=exp(-1+sin(2*pi*x5)) # inefficiency parameter

y<-rcomper(n=N, mu=mu, sigma_v=sigma_v, sigma_u=sigma_u, s=s, distr="normhnorm")
dat<-data.frame(y, x1, x2, x3, x4, x5)

# Write formulae for parameters
mu_formula<-y~x1+x2+I(x2^2)+s(x3, bs="ps")
sigma_v_formula<-~1+x4
sigma_u_formula<-~1+s(x5, bs="ps")

# Fit model
model<-dsfa(formula=list(mu_formula, sigma_v_formula, sigma_u_formula),
             data=dat, family=comper(s=s, distr="normhnorm"), optimizer = c("efs"))

# Estimate efficiency
efficiency(model, type="jondrow")
efficiency(model, type="battese")
```

**elasticity**

**Description**

Calculates and plots the elasticity of a smooth function.

**Usage**

`elasticity(object, select = NULL, plot = TRUE, se = TRUE)`
Arguments

- **object**: fitted mgcv object with family `comper()` or `comper_mv()`.
- **select**: specifying the smooth function for which the elasticity is calculated. If `term=NULL` the elasticities for all smooths of $\mu$ are returned (excluding random and spatial effects).
- **plot**: logical; if TRUE, plots the elasticities. If FALSE, returns the average elasticity.
- **se**: logical; if TRUE, adds standard errors to the plot of elasticities.

Details

Calculates the marginal product for parametric terms. For smooth terms the average of the derivative is calculated.

Value

If plot is TRUE, plots the elasticities specified in select of the provided object. If plot is FALSE returns a named vector of the elasticity of the provided inputs.

References


Examples

```r
# Set seed, sample size and type of function
set.seed(1337)
N=500 # Sample size
s=-1 # Set to production function

# Generate covariates
x1<-runif(N,-1,1); x2<-runif(N,-1,1); x3<-runif(N,-1,1)
x4<-runif(N,-1,1); x5<-runif(N,-1,1)

# Set parameters of the distribution
mu=2+0.75*x1+0.4*x2+0.6*x2^2+6*log(x3+2)^(1/4) # production function parameter
sigma_v=exp(-1.5+0.75*x4) # noise parameter
sigma_u=exp(-1+sin(2*pi*x5)) # inefficiency parameter

y<-rcomper(n=N, mu=mu, sigma_v=sigma_v, sigma_u=sigma_u, s=s, distr="normhnorm")
dat<-data.frame(y, x1, x2, x3, x4, x5)
```
# Write formulae for parameters
mu_formula<-y~x1+x2+I(x2^2)+s(x3, bs="ps")
sigma_v_formula<-~1+x4
sigma_u_formula<-~1+s(x5, bs="ps")

# Fit model
model<-dsfa(formula=list(mu_formula, sigma_v_formula, sigma_u_formula),
             data=dat, family=comper(s=s, distr="norm~norm"), optimizer = c("efs"))

# Get elasticities
elasticity(model, plot=TRUE)

ind2joint

## Independent to joint function

### Description
Combines multiple derivs objects into a single derivs object.

### Usage

```r
ind2joint(f_list, tri_f_list, tri_h_list, deriv_order)
```

### Arguments

- **f_list**: list of derivs objects of length \( M \), e.g. \(\text{list}(f_1(\cdot), f_2(\cdot), \ldots, f_M(\cdot))\)
- **tri_f_list**: list of length \( K \) trind_generator objects, the \( k \)th element corresponds to \( k \)th derivs object.
- **tri_h_list**: list of length \( K \) trind_generator objects, the \( k \)th element corresponds to a derivs object with \( k \cdot (k + 1)/2 \) parameters.
- **deriv_order**: integer; maximum order of derivative. Available are 0, 2 and 4.

### Details

Let \( f_m \) be a function defined in \([\text{trind()}]\), where \( m \in 1, \ldots, M \). Define \( h((x_{n1}, x_{n2}, \ldots, x_{nK})) = (f_1(x_{n1}), f_2(x_{n2}), \ldots, f_M(x_{nK})). \) In order to get the derivatives of \( h(\cdot) \) w.r.t all parameters \( x_{nk} \), the independent functions are combined. For more details see \([\text{trind()}]\) and \([\text{trind_generator()}]\).

### Value

Returns a derivs object.

### See Also

Other derivs: \texttt{chainrule()}, \texttt{derivs_transform()}, \texttt{differencerule()}, \texttt{list2derivs()}, \texttt{productrule()}, \texttt{quotientrule()}, \texttt{sumrule()}, \texttt{trind_generator()}.
Examples

```r
A <- matrix(c(1:9)/10, ncol=1)
A_derivs <- list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=4)
B_derivs <- transform(A, type="exp", par=0, deriv_order=4)
ind2joint(list(A_derivs, B_derivs),
  list(trind_generator(1), trind_generator(1)),
  list(trind_generator(1), trind_generator(1+1)), 4)
```

Description

Transforms a list of matrices d0, d1, d2, d3, d4 to a derivs object.

Usage

```r
list2derivs(f, deriv_order)
```

Arguments

- `f` list of matrices; d0, d1, d2, d3, d4
- `deriv_order` integer; maximum order of derivative. Available are 0, 2 and 4.

Value

Mostly internal function. Returns an object of class derivs For more details see [trind()] and [trind_generator()].

See Also

Other derivs: `chainrule()`, `derivs_transform()`, `differencerule()`, `ind2joint()`, `productrule()`, `quotientrule()`, `sumrule()`, `trind_generator()`, `trind()`

Examples

```r
A <- matrix(c(1:9)/10, ncol=3)
list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=4)
```
Description

Dataset of the National Bureau of Economic Research (NBER) and U.S. Census Bureau’s Center for Economic Studies (CES). It contains information on the annual industry-level from 1958-2016 of the US.

Usage

manuf

Format

manuf is a data frame with 6188 rows and 7 columns:

- **naics**  NAICS 2012 6-digit industry code
- **year**  Year from 2000 to 2016
- **Y**  Total value of shipments in millions of 2012 dollars
- **K**  Real capital stock in millions of 2012 dollars
- **L**  Production worker hours in millions
- **M**  Total cost of materials in millions of 2012 dollars
- **I**  New capital spending in millions of 2012 dollars

Details

This is a subset of the data which contains data from 2000-2016 on output, employment, materials, investment and capital stocks.

Source

<https://www.nber.org/research/data/nber-ces-manufacturing-industry-database>

References

mom2par  

Moments to Parameters

Description

Calculates the parameters of composed-error distribution based on the provided moments.

Usage

mom2par(mean = 0, sd = 1, skew = 0, s = -1, distr = "normhnorm")

Arguments

mean  numeric vector of means.
sd    numeric vector of standard deviations. Must be positive.
skew  numeric vector of skewness. s*skew must be positive.
s     integer; s = -1 for production and s = 1 for cost function.
distr string; determines the distribution:
"normhnorm", Normal-halfnormal distribution
"normexp", Normal-exponential distribution

Details

See dcomper for details of the distribution. For the inverse transformation see par2mom.

Value

Returns a matrix where the first column corresponds to $\mu$, the second to $\sigma_V$ and the third to $\sigma_U$.

Examples

mom2par(mean=0, sd=1, skew=-0.5, s=-1, distr="normhnorm")
mom2par(mean=0, sd=1, skew=-1, s=-1, distr="normexp")
par2mom  

Parameter to Moments

Description

Calculates the moments of composed-error distribution based on the provided parameters.

Usage

par2mom(mu = 0, sigma_v = 1, sigma_u = 1, s = -1, distr = "normhnorm")

Arguments

mu  numeric vector of $\mu$.

sigma_v  numeric vector of $\sigma_V$. Must be positive.

sigma_u  numeric vector of $\sigma_U$. Must be positive.

s  integer; $s = -1$ for production and $s = 1$ for cost function.

distr  string; determines the distribution:

'normhnorm', Normal-halfnormal distribution

'normexp', Normal-exponential distribution

Details

See dcomper for details of the distribution. For the inverse transformation see mom2par.

Value

Returns a matrix where the first column corresponds to the mean, the second to the standard deviation and the third to the skewness.

Examples

par2mom(mu=0, sigma_v=1, sigma_u=1, s=-1, distr="normhnorm")
par2mom(mu=0, sigma_v=1, sigma_u=1, s=-1, distr="normexp")
Description

Productrule for derivs objects.

Usage

productrule(f_list, tri, deriv_order)

Arguments

- **f_list**: list of derivs objects of length \( M \), e.g. \( list(f_1(\cdot), f_2(\cdot), ..., f_M(\cdot)) \)
- **tri**: list; created by the function \( \text{trind_generator()} \).
- **deriv_order**: integer; maximum order of derivative. Available are 0, 2 and 4.

Details

Let \( f_m \) be a function defined in \( \text{trind()} \), where \( m \in 1, ..., M \). Define \( h(x_{n1}, x_{n2}, ..., x_{nK}) = f_1(\cdot) \cdot f_2(\cdot) ... \cdot f_M(x_{n1}, x_{n2}, ..., x_{nK}) \). In order to get the derivatives of \( h(\cdot) \) w.r.t all parameters \( x_{nk} \), the productrule is applied. For more details see \( \text{trind()} \) and \( \text{trind_generator()} \).

Value

Returns an object of class derivs for the function \( h(\cdot) \).

See Also

Other derivs: \( \text{chainrule()}, \text{derivs_transform()}, \text{differencerule()}, \text{ind2joint()}, \text{list2derivs()}, \text{quotientrule()}, \text{sumrule()}, \text{trind_generator()}, \text{trind()} \)

Examples

```r
A<-matrix(c(1:9)/10, ncol=1)
A_derivs<-list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=2)
B_derivs<-derivs_transform(A, type="inv", par=0, trind_generator(1), deriv_order=2)
productrule (list(A_derivs, B_derivs), trind_generator(1), deriv_order=2) #identity
```
Description
Quotientrule for derivs objects.

Usage
quotientrule(f_list, tri, deriv_order)

Arguments
- f_list: list of derivs objects of length \( M \), e.g. \( \text{list}(f_1(\cdot), f_2(\cdot), ..., f_M(\cdot)) \)
- tri: list; created by the function \([\text{trind\_generator()}]\).
- deriv_order: integer; maximum order of derivative. Available are 0, 2 and 4.

Details
Let \( f_m \) be a function defined in \([\text{trind()}]\), where \( m \in 1, ..., M \). Define \( h((x_{n_1}, x_{n_2}, ..., x_{n_K})) = \frac{f_1(\cdot)}{f_2(\cdot)} ... \frac{f_M(\cdot)}{f_M(x_{n_1}, x_{n_2}, ..., x_{n_K})} \). In order to get the derivatives of \( h(\cdot) \) w.r.t all parameters \( x_{n_k} \), the quotientrule is applied. For more details see \([\text{trind()}]\) and \([\text{trind\_generator()}]\). The values of the derivs objects must be positive. Numerically not precise, but included for reasons of completeness.

Value
Returns an object of class derivs for the function \( h(\cdot) \).

See Also
Other derivs: \text{chainrule()}, \text{derivs\_transform()}, \text{differencerule()}, \text{ind2joint()}, \text{list2derivs()}, \text{productrule()}, \text{sumrule()}, \text{trind\_generator()}, \text{trind()}

Examples
\begin{verbatim}
A <- matrix(c(1:9)/10, ncol=1)
A_derivs <- list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=2)
B_derivs <- derivs_transform(A, type="inv", par=0, trind_generator(1), deriv_order=2)
quotientrule(list(A_derivs, B_derivs), trind_generator(1), deriv_order=2) # A/(1/A) = A^2
\end{verbatim}
sumrule

Description

Sum rule for derivs objects.

Usage

sumrule(f_list, tri, deriv_order)

Arguments

f_list list of derivs objects of length \( M \), e.g. \( \text{list}(f_1(\cdot), f_2(\cdot),..., f_M(\cdot)) \)

tri list; created by the function \([\text{trind}\_\text{generator}()]\).

deriv_order integer; maximum order of derivative. Available are 0, 2 and 4.

Details

Let \( f_m \) be a function defined in \([\text{trind}()]\), where \( m \in 1,..., M \). Define \( h((x_{n1}, x_{n2},..., x_{nK})) = f_1(\cdot) + f_2(\cdot) +...+ f_M(x_{n1}, x_{n2},..., x_{nK})) \). In order to get the derivatives of \( h(\cdot) \) w.r.t all parameters \( x_{nk} \), the sumrule is applied. For more details see \([\text{trind}()]\) and \([\text{trind}\_\text{generator}()]\).

Value

Returns an object of class derivs for the function \( h(\cdot) \).

See Also

Other derivs: \([\text{chainrule}()\), \([\text{derivs}\_\text{transform}()\), \([\text{differencerule}()\), \([\text{ind2joint}()\), \([\text{list2derivs}()\), \([\text{productrule}()\), \([\text{quotientrule}()\), \([\text{trind}\_\text{generator}()\), \([\text{trind}()\)

Examples

A<-matrix(c(1:9)/10, ncol=1)
A_derivs<-list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=4)
sumrule(list(A_derivs, A_derivs), trind_generator(1), deriv_order=4) #equal to 2*A_derivs
Description

Transforms a matrix via the specified function.

Usage

transform(x, type, par, deriv_order)

Arguments

x numeric matrix to be transformed.
type string, specifies the transformation function. Available are:
- 'identity': \( f(x) = x \).
- 'exp': \( f(x) = \exp\{x\} \).
- 'log': \( f(x) = \log\{x\} \).
- 'glogit': \( f(x) = \log\{(-x + \text{min})/(x - \text{max})\}, \) where \( \text{par} = (\text{min}, \text{max}) \).
- 'glogitinv': \( f(x) = \exp\{x\}\cdot(\text{max} + \text{min})/(1 + \exp\{x\}) \), where \( \text{par} = (\text{min}, \text{max}) \).
- 'inv': \( f(x) = \frac{1}{x} \).
- 'pnorm': \( f(x) = \Phi(x) \).
- 'qnorm': \( f(x) = \Phi^{-1}(x) \).
- 'mexp': \( f(x) = -\exp\{x\} \).
- 'zeta': \( f(x) = \log\{|2\cdot\Phi(x)|\} \).
- 'constant': \( f(x) = c \).
- 'chainrule_utility': \( f(x) = f'(x) = f''(x) = f'''(x) = f''''(x) \).
- 'onemx': \( 1 - x \).

par numeric vector, additional parameters, e.g. min and max for glogit.
deriv_order integer; maximum order of derivative. Available are 0, 2 and 4.

Details

Takes the numeric matrix x as an input for the function specified by type and evaluates it together with the derivatives.

Value

Returns an object of class derivs.

Examples

A<-matrix(c(1:9)/10, ncol=3)
A_mat<-list2derivs(list(A, A^0, A^2, A^3, A^4), deriv_order=4)
transform(x=transform(x = A, type="exp", par=0, deriv_order=4), type="log", deriv_order=4, par = 0)
Description

Provides the column index of the required derivative for the specified order of a derivs object.

Usage

trind(tri, part_deriv_var)

Arguments

tri list; created by the function [trind_generator()].

part_deriv_var integer vector; specifies $\frac{\partial^i f()}{\partial x_n_1 \partial x_n_1 \ldots \partial x_n_J}$. The length of the vector is denoted as $J$ and determines the order of the partial derivatives with maximum four. The element $i_j \in \{0, \ldots, K - 1\}$ specifies the variable with respect to which the derivative is taken, where $j \in \{1, \ldots, J\}$. The order corresponds to the order of derivatives. For example $c(0, 0, 1, 2)$ is equal to $\frac{\partial^4 f()}{\partial x_n_1 \partial x_n_1 \partial x_n_1 \partial x_n_3}$. See details for more information.

Details

Let $f : \mathbb{R}^K \rightarrow \mathbb{R}^L, (x_n_1, x_n_2, \ldots, x_n_K) \rightarrow f(x_n_1, x_n_2, \ldots, x_n_K)$ be differentiable up to order four w.r.t all parameters $x_n_k$, where $k \in \{1, \ldots, K\}$ and $n \in \{1, \ldots, N\}$. Then a derivs class object is a numeric matrix with $N$ rows and $L$ columns. $N$ is the length of the input vectors. Further, it has the following attributes:

1. ‘d1’: a numeric matrix of the first derivatives w.r.t all parameters, where the $n$th row corresponds to: $(\frac{\partial f()}{\partial x_n_1}, \frac{\partial f()}{\partial x_n_1}, \ldots, \frac{\partial f()}{\partial x_n_K})$
2. ‘d2’: a numeric matrix of the second derivatives w.r.t all parameters, where the $n$th row corresponds to: $(\frac{\partial^2 f()}{\partial x_n_1 \partial x_n_1}, \frac{\partial^2 f()}{\partial x_n_1 \partial x_n_2}, \ldots, \frac{\partial^2 f()}{\partial x_n_K \partial x_n_K})$
3. ‘d3’: a numeric matrix of the third derivatives w.r.t all parameters, where the $n$th row corresponds to: $(\frac{\partial^3 f()}{\partial x_n_1 \partial x_n_1 \partial x_n_1}, \frac{\partial^3 f()}{\partial x_n_1 \partial x_n_1 \partial x_n_2}, \ldots, \frac{\partial^3 f()}{\partial x_n_K \partial x_n_K \partial x_n_K})$
4. ‘d4’: a numeric matrix of the fourth derivatives w.r.t all parameters, where the $n$th row corresponds to: $(\frac{\partial^4 f()}{\partial x_n_1 \partial x_n_1 \partial x_n_1 \partial x_n_1}, \frac{\partial^4 f()}{\partial x_n_1 \partial x_n_1 \partial x_n_1 \partial x_n_2}, \ldots, \frac{\partial^4 f()}{\partial x_n_K \partial x_n_K \partial x_n_K \partial x_n_K})$

The function trind() provides the index for the corresponding derivatives. The derivs class object allows for a modular system which can be easily extended and is faster than numerical derivatives. The advantage compared to analytical derivatives provided by ‘mathematica’ or deriv() is that asymptotics and approximations can be used for individual parts. Handwritten derivatives can be tedious at times and may be prone to errors. Thus, the derivs class object can be used by lazy users. Mainly intended for internal use.
trind_generator

Value

Integer, the index for a derivs object.

See Also

Other derivs: chainrule(), derivs_transform(), differencerule(), ind2joint(), list2derivs(), productrule(), quotientrule(), sumrule(), trind_generator()

Examples

tri=trind_generator(3)
trind(tri, c(2,1))

Description

Generates index matrices for upper triangular storage up to order four.

Usage

trind_generator(K)

Arguments

K integer; determines the number of parameters.

Details

Useful when working with higher order derivatives, which generate symmetric arrays. Mainly intended for internal use. Similar to 'mgcv::trind.generator'. Mostly internal function.

Value

Returns a list with index matrices for the first to fourth derivative, which can be accessed via the function [trind()]. The numerical vectors i_start and i_end hold the starting and ending indexes, which are required by [trind()] for derivatives greater than two.

See Also

Other derivs: chainrule(), derivs_transform(), differencerule(), ind2joint(), list2derivs(), productrule(), quotientrule(), sumrule(), trind()
Examples

```
tri <- trind_generator(3)
tri_mgcve <- mgcv::trind.generator(3)

for(i in 1:3){
  print(i == trind(tri, part_deriv_var=c(i)-1)+1)
  for(j in i:3){
    print(tri_mgcve$i2[i,j] == trind(tri, part_deriv_var=c(i,j)-1)+1)
    for(k in j:3){
      print(tri_mgcve$i3[i,j,k] == trind(tri, part_deriv_var=c(i,j,k)-1)+1)
      for(l in k:3){
        print(tri_mgcve$i4[i,j,k,l] == trind(tri, part_deriv_var=c(i,j,k,l)-1)+1)
      }
    }
  }
}
```
Index

* copula
  cop, 13
dcop, 20
delta_bounds, 22

* datasets
  BurkinaFarms, 2
  BurkinaFarms_polys, 4
  manuf, 39

* derivs
  chainrule, 5
derivs_transform, 23
differencerule, 24
ind2joint, 37
list2derivs, 38
productrule, 42
quotientrule, 43
sumrule, 44
trind, 46
trind_generator, 47

* distribution
  dcomper, 15
dcomper_mv, 17
dnormexp, 25
dnormhnorm, 27

anova.gam, 30
BurkinaFarms, 2
BurkinaFarms_polys, 4
cdf2quantile, 4
chainrule, 5, 24, 25, 37, 38, 42–44, 47
comper, 6, 30, 31
comper_mv, 9, 30, 31
cop, 13, 22, 23, 31
dcomper, 15, 20, 27, 29–31, 40, 41
dcomper_mv, 17, 17, 27, 29, 31
dcop, 14, 20, 23, 31
delta_bounds, 14, 22, 22
deriv(), 46
derivs_transform, 5, 23, 25, 37, 38, 42–44, 47
differencerule, 5, 24, 24, 37, 38, 42–44, 47
dnormexp, 16, 17, 20, 25, 29, 30
dnormhnorm, 16, 17, 20, 27, 27, 30
dsfa, 29
efficiency, 30, 34
elasticity, 30, 35
formula.gam, 31
gam, 30, 31
gam(), 6, 10, 14
gam.models, 31
gamObject, 30
ind2joint, 5, 24, 25, 37, 38, 42–44, 47
list2derivs, 5, 24, 25, 37, 38, 42–44, 47
manuf, 39
mom2par, 40, 41
mrf, 30
normexp (dnormexp), 25
normhnorm (dnormhnorm), 27
p.spline, 30
par2mom, 40, 41
pcomper (dcomper), 15
pcomper_mv (dcomper_mv), 17
pcop (dcop), 20
pcopula, 21
plot.gam, 30
predict.gam, 30
pnormexp (dnormexp), 25
pnormhnorm (dnormhnorm), 27
predict.gam, 30
productrule, 5, 24, 25, 37, 38, 42, 43, 44, 47
qcomper (dcomper), 15
qnormexp (dnormexp), 25
qnormhnorm (dnormhnorm), 27
quotientrule, 5, 24, 25, 37, 38, 42, 43, 44, 47

random.effects, 30
rcomper (dcomper), 15
rcomper_mv (dcomper_mv), 17
rcop (dcop), 20
rCopula, 21
residuals.gam, 30
rnormexp (dnormexp), 25
rnormhnorm (dnormhnorm), 27
s, 30
smooth.terms, 30
summary.gam, 30
sumrule, 5, 24, 25, 37, 38, 42, 43, 44, 47
tprs, 30
transform, 45
trind, 5, 21, 24–26, 28, 37, 38, 42–44, 46, 47
trind_generator, 5, 16, 21, 24–26, 28, 37, 38, 42–44, 47, 47