Package ‘snfa’

November 20, 2018

Title Smooth Non-Parametric Frontier Analysis

Version 0.0.1

Description Fitting of non-parametric production frontiers for use in efficiency analysis.
Methods are provided for both a smooth analogue of Data Envelopment Analysis (DEA) and a non-parametric analogue of Stochastic Frontier Analysis (SFA). Frontiers are constructed for multiple inputs and a single output using constrained kernel smoothing as in Racine et al. (2009), which allow for the imposition of monotonicity and concavity constraints on the estimated frontier.

Maintainer Taylor McKenzie <tkmckenzie@gmail.com>

Depends R (>= 3.5.0)

Imports abind (>= 1.4.5), ggplot2 (>= 3.1.0), prodlim (>= 2018.4.18), quadprog (>= 1.5.5), Rdpack (>= 0.10.1), rootSolve (>= 1.7)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

RdMacros Rdpack

NeedsCompilation no

Author Taylor McKenzie [aut, cre]

R topics documented:

`allocative.efficiency` *Allocative efficiency estimation*

Description

Fits frontier to data and estimates technical and allocative efficiency

Usage

```
allocative.efficiency(X, y, X.price, y.price, X.constrained = NA,
  H.inv = NA, H.mult = 1, model = "br", method = "u",
  scale.constraints = TRUE)
```

Arguments

`X`	Matrix of inputs
`y`	Vector of outputs
`X.price`	Matrix of input prices
`y.price`	Vector of output prices
`X.constrained`	Matrix of inputs where constraints apply
`H.inv`	Inverse of the smoothing matrix (must be positive definite); defaults to rule of thumb
`H.mult`	Scaling factor for rule of thumb smoothing matrix
`model`	Type of frontier to use; "br" for boundary regression, "sf" for stochastic frontier
`method`	Constraints to apply; "u" for unconstrained, "m" for monotonically increasing, and "mc" for monotonically increasing and concave
`scale.constraints`	Boolean, whether to scale constraints by their average value, can help with convergence

Details

This function estimates allocative inefficiency using the methodology in McKenzie (2018). The estimation process is a non-parametric analogue of Schmidt and Lovell (1979). First, the frontier is fit using either a boundary regression or stochastic frontier as in Racine et al. (2009), from which technical efficiency is estimated. Then, gradients and price ratios are computed for each observation and compared to determine the extent of misallocation. Specifically, log-overallocation is computed as

$$\log\left(\frac{w_i^j}{p_i}\right) - \log\left(\phi_i \frac{\partial f(x_i)}{\partial x^j}\right),$$

where ϕ_i is the efficiency of observation i, $\partial f(x_i)/\partial x^j$ is the marginal productivity of input j at observation i, w_i^j is the cost of input j for observation i, and p_i is the price of output for observation i.

Value

Returns a list with the following elements

`y.fit`	Estimated value of the frontier at X.fit
`gradient.fit`	Estimated gradient of the frontier at X.fit
`technical.efficiency`	Estimated technical efficiency
`log.overallocation`	Estimated log-overallocation of each input for each observation
`X.eval`	Matrix of inputs used for fitting
`X.constrained`	Matrix of inputs where constraints apply
`H.inv`	Inverse smoothing matrix used in fitting
`method`	Method used to fit frontier
`scaling.factor`	Factor by which constraints are multiplied before quadratic programming

References

Aigner D, Lovell CK, Schmidt P (1977). "Formulation and estimation of stochastic frontier production function models." *Journal of econometrics*, **6**(1), 21–37.

McKenzie T (2018). "Semi-Parametric Estimation of Allocative Inefficiency Using Smooth Non-Parametric Frontier Analysis." Working Paper.

Racine JS, Parmeter CF, Du P (2009). "Constrained nonparametric kernel regression: Estimation and inference." Working paper.

Schmidt P, Lovell CK (1979). "Estimating technical and allocative inefficiency relative to stochastic production and cost frontiers." *Journal of econometrics*, **9**(3), 343–366.

Examples

```
data(USMacro)

USMacro <- USMacro[complete.cases(USMacro),]

#Extract data
X <- as.matrix(USMacro[,c("K", "L")])
y <- USMacro$Y

X.price <- as.matrix(USMacro[,c("K.price", "L.price")])
y.price <- rep(1e9, nrow(USMacro)) #Price of $1 billion of output is $1 billion

#Run model
efficiency.model <- allocative.efficiency(X, y,
                                         X.price, y.price,
                                         X.constrained = X,
                                         model = "br",
                                         method = "mc")

#Plot technical/allocative efficiency over time
library(ggplot2)
```

```
technical.df <- data.frame(Year = USMacro$Year,
                          Efficiency = efficiency.model$technical.efficiency)

ggplot(technical.df, aes(Year, Efficiency)) +
  geom_line()

allocative.df <- data.frame(Year = rep(USMacro$Year, times = 2),
                         log.overallocation = c(efficiency.model$log.overallocation[,1],
                                              efficiency.model$log.overallocation[,2]),
                            Variable = rep(c("K", "L"), each = nrow(USMacro)))

ggplot(allocative.df, aes(Year, log.overallocation)) +
  geom_line(aes(color = Variable))

#Estimate average overallocation across sample period
lm.model <- lm(log.overallocation ~ 0 + Variable, allocative.df)
summary(lm.model)
```

`fit.boundary` *Multivariate smooth boundary fitting with additional constraints*

Description

Fits boundary of data with kernel smoothing, imposing monotonicity and/or concavity constraints.

Usage

```
fit.boundary(X.eval, y.eval, X.bounded, y.bounded, X.constrained = NA,
  X.fit = NA, y.fit.observed = NA, H.inv = NA, H.mult = 1,
  method = "u", scale.constraints = TRUE)
```

Arguments

`X.eval`	Matrix of inputs used for fitting
`y.eval`	Vector of outputs used for fitting
`X.bounded`	Matrix of inputs where bounding constraints apply
`y.bounded`	Vector of outputs where bounding constraints apply
`X.constrained`	Matrix of inputs where monotonicity/concavity constraints apply
`X.fit`	Matrix of inputs where curve is fit; defaults to X.constrained
`y.fit.observed`	Vector of outputs corresponding to observations in X.fit; used for efficiency calculation
`H.inv`	Inverse of the smoothing matrix (must be positive definite); defaults to rule of thumb
`H.mult`	Scaling factor for rule of thumb smoothing matrix
`method`	Constraints to apply; "u" for unconstrained, "m" for monotonically increasing, and "mc" for monotonically increasing and concave
`scale.constraints`	Boolean, whether to scale constraints by their average value, can help with convergence

Details

This method fits a smooth boundary of the data (with all data points below the boundary) while imposing specified monotonicity and concavity constraints. The procedure is derived from Racine et al. (2009), which develops kernel smoothing methods with bounding, monotonicity and concavity constraints. Specifically, the smoothing procedure involves finding optimal weights for a Nadaraya-Watson estimator of the form

$$\hat{y} = m(x) = \sum_{i=1}^{N} p_i A(x, x_i) y_i,$$

where x are inputs, y are outputs, p are weights, subscripts index observations, and

$$A(x, x_i) = \frac{K(x, x_i)}{\sum_{h=1}^{N} K(x, x_h)}$$

for a kernel K. This method uses a multivariate normal kernel of the form

$$K(x, x_h) = \exp\left(-\frac{1}{2}(x - x_h)' H^{-1} (x - x_h)\right),$$

where H is a bandwidth matrix. Bandwidth selection is performed via Silverman's (1986) rule-of-thumb, in the function `H.inv.select`.

Optimal weights $\hat{p}$ are selected by solving the quadratic programming problem

$$\min_p \quad -\mathbf{1}'p + \frac{1}{2}p'p.$$

This method always imposes bounding constraints as specified points, given by

$$m(x_i) - y_i = \sum_{h=1}^{N} p_h A(x_i, x_h) y_h - y_i \geq 0 \quad \forall i.$$

Additionally, monotonicity constraints of the following form can be imposed at specified points:

$$\frac{\partial m(x)}{\partial x^j} = \sum_{h=1}^{N} p_h \frac{\partial A(x, x_h)}{\partial x^j} y_h \geq 0 \quad \forall x, j,$$

where superscripts index inputs. Finally concavity constraints of the following form can also be imposed using Afriat's (1967) conditions:

$$m(x) - m(z) \leq \nabla_x m(z) \cdot (x - z) \quad \forall x, z.$$

The gradient of the frontier at a point x is given by

$$\nabla_x m(x) = \sum_{i=1}^{N} \hat{p}_i \nabla_x A(x, x_i) y_i,$$

where $\hat{p}_i$ are estimated weights.

Value

Returns a list with the following elements

`y.fit`	Estimated value of the frontier at X.fit
`gradient.fit`	Estimated gradient of the frontier at X.fit
`efficiency`	Estimated efficiencies of y.fit.observed
`solution`	Boolean; TRUE if frontier successfully estimated
`X.eval`	Matrix of inputs used for fitting
`X.constrained`	Matrix of inputs where monotonicity/concavity constraints apply
`X.fit`	Matrix of inputs where curve is fit
`H.inv`	Inverse smoothing matrix used in fitting
`method`	Method used to fit frontier
`scaling.factor`	Factor by which constraints are multiplied before quadratic programming

References

Racine JS, Parmeter CF, Du P (2009). “Constrained nonparametric kernel regression: Estimation and inference.” Working paper.

Examples

```
data(univariate)

#Set up data for fitting

X <- as.matrix(univariate$x)
y <- univariate$y

N.fit <- 100
X.fit <- as.matrix(seq(min(X), max(X), length.out = N.fit))

#Reflect data for fitting
reflected.data <- reflect.data(X, y)
X.eval <- reflected.data$X
y.eval <- reflected.data$y

#Fit frontiers
frontier.u <- fit.boundary(X.eval, y.eval,
                           X.bounded = X, y.bounded = y,
                           X.constrained = X.fit,
                           X.fit = X.fit,
                           method = "u")

frontier.m <- fit.boundary(X.eval, y.eval,
                           X.bounded = X, y.bounded = y,
                           X.constrained = X.fit,
                           X.fit = X.fit,
                           method = "m")

frontier.mc <- fit.boundary(X.eval, y.eval,
                            X.bounded = X, y.bounded = y,
                            X.constrained = X.fit,
                            X.fit = X.fit,
```

```
                              method = "mc")

#Plot frontier
library(ggplot2)

frontier.df <- data.frame(X = rep(X.fit, times = 3),
                          y = c(frontier.u$y.fit, frontier.m$y.fit, frontier.mc$y.fit),
                          model = rep(c("u", "m", "mc"), each = N.fit))

ggplot(univariate, aes(X, y)) +
  geom_point() +
  geom_line(data = frontier.df, aes(color = model))

#Plot slopes
slope.df <- data.frame(X = rep(X.fit, times = 3),
                       slope = c(frontier.u$gradient.fit,
                                 frontier.m$gradient.fit,
                                 frontier.mc$gradient.fit),
                       model = rep(c("u", "m", "mc"), each = N.fit))

ggplot(slope.df, aes(X, slope)) +
  geom_line(aes(color = model))
```

`fit.mean` *Kernel smoothing with additional constraints*

Description

Fits conditional mean of data with kernel smoothing, imposing monotonicity and/or concavity constraints.

Usage

```
fit.mean(X.eval, y.eval, X.constrained = NA, X.fit = NA, H.inv = NA,
  H.mult = 1, method = "u", scale.constraints = TRUE)
```

Arguments

`X.eval`	Matrix of inputs used for fitting
`y.eval`	Vector of outputs used for fitting
`X.constrained`	Matrix of inputs where constraints apply
`X.fit`	Matrix of inputs where curve is fit; defaults to X.constrained
`H.inv`	Inverse of the smoothing matrix (must be positive definite); defaults to rule of thumb
`H.mult`	Scaling factor for rule of thumb smoothing matrix
`method`	Constraints to apply; "u" for unconstrained, "m" for monotonically increasing, and "mc" for monotonically increasing and concave
`scale.constraints`	Boolean, whether to scale constraints by their average value, can help with convergence

Details

This method uses kernel smoothing to fit the mean of the data while imposing specified monotonicity and concavity constraints. The procedure is derived from Racine et al. (2009), which develops kernel smoothing methods with bounding, monotonicity and concavity constraints. Specifically, the smoothing procedure involves finding optimal weights for a Nadaraya-Watson estimator of the form

$$\hat{y} = m(x) = \sum_{i=1}^{N} p_i A(x, x_i) y_i,$$

where x are inputs, y are outputs, p are weights, subscripts index observations, and

$$A(x, x_i) = \frac{K(x, x_i)}{\sum_{h=1}^{N} K(x, x_h)}$$

for a kernel K. This method uses a multivariate normal kernel of the form

$$K(x, x_h) = \exp\left(-\frac{1}{2}(x - x_h)' H^{-1}(x - x_h)\right),$$

where H is a bandwidth matrix. Bandwidth selection is performed via Silverman's (1986) rule-of-thumb, in the function `H.inv.select`.

Optimal weights $\hat{p}$ are selected by solving the quadratic programming problem

$$\min_p \quad -\mathbf{1}'p + \frac{1}{2}p'p.$$

Monotonicity constraints of the following form can be imposed at specified points:

$$\frac{\partial m(x)}{\partial x^j} = \sum_{h=1}^{N} p_h \frac{\partial A(x, x_h)}{\partial x^j} y_h \geq 0 \quad \forall x, j,$$

where superscripts index inputs. Finally concavity constraints of the following form can also be imposed using Afriat's (1967) conditions:

$$m(x) - m(z) \leq \nabla_x m(z) \cdot (x - z) \quad \forall x, z.$$

The gradient of the estimated curve at a point x is given by

$$\nabla_x m(x) = \sum_{i=1}^{N} \hat{p}_i \nabla_x A(x, x_i) y_i,$$

where $\hat{p}_i$ are estimated weights.

Value

Returns a list with the following elements

`y.fit`	Estimated value of the frontier at X.fit
`gradient.fit`	Estimated gradient of the frontier at X.fit
`solution`	Boolean; TRUE if frontier successfully estimated
`X.eval`	Matrix of inputs used for fitting
`X.constrained`	Matrix of inputs where constraints apply
`X.fit`	Matrix of inputs where curve is fit
`H.inv`	Inverse smoothing matrix used in fitting
`method`	Method used to fit frontier
`scaling.factor`	Factor by which constraints are multiplied before quadratic programming

References

Racine JS, Parmeter CF, Du P (2009). "Constrained nonparametric kernel regression: Estimation and inference." Working paper.

Examples

```
data(USMacro)

USMacro <- USMacro[complete.cases(USMacro),]

#Extract data
X <- as.matrix(USMacro[,c("K", "L")])
y <- USMacro$Y

#Reflect data for fitting
reflected.data <- reflect.data(X, y)
X.eval <- reflected.data$X
y.eval <- reflected.data$y

#Fit frontier
fit.mc <- fit.mean(X.eval, y.eval,
                   X.constrained = X,
                   X.fit = X,
                   method = "mc")

#Plot input productivities over time
library(ggplot2)
plot.df <- data.frame(Year = rep(USMacro$Year, times = 2),
                      Elasticity = c(fit.mc$gradient.fit[,1] * X[,1] / y,
                                     fit.mc$gradient.fit[,2] * X[,2] / y),
                      Variable = rep(c("Capital", "Labor"), each = nrow(USMacro)))

ggplot(plot.df, aes(Year, Elasticity)) +
  geom_line() +
  facet_grid(Variable ~ ., scales = "free_y")
```

fit.sf *Non-parametric stochastic frontier*

Description

Fits stochastic frontier of data with kernel smoothing, imposing monotonicity and/or concavity constraints.

Usage

```
fit.sf(X, y, X.constrained = NA, H.inv = NA, H.mult = 1,
  method = "u", scale.constraints = TRUE)
```

Arguments

X	Matrix of inputs
y	Vector of outputs
X.constrained	Matrix of inputs where constraints apply
H.inv	Inverse of the smoothing matrix (must be positive definite); defaults to rule of thumb
H.mult	Scaling factor for rule of thumb smoothing matrix
method	Constraints to apply; "u" for unconstrained, "m" for monotonically increasing, and "mc" for monotonically increasing and concave
scale.constraints	Boolean, whether to scale constraints by their average value, can help with convergence

Details

This method fits non-parametric stochastic frontier models. The data-generating process is assumed to be of the form

$$\ln y_i = \ln f(x_i) + v_i - u_i,$$

where y_i is the ith observation of output, f is a continuous function, x_i is the ith observation of input, v_i is a normally-distributed error term ($v_i \sim N(0, \sigma_v^2)$), and u_i is a normally-distributed error term truncated below at zero ($u_i \sim N^+(0, \sigma_u)$). Aigner et al. developed methods to decompose $\varepsilon_i = v_i - u_i$ into its basic components.

This procedure first fits the mean of the data using `fit.mean`, producing estimates of output $\hat{y}$. Log-proportional errors are calculated as

$$\varepsilon_i = \ln(y_i/\hat{y}_i).$$

Following Aigner et al. (1977), parameters of one- and two-sided error distributions are estimated via maximum likelihood. First,

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N} \varepsilon_i^2.$$

Then, $\hat{\lambda}$ is estimated by solving

$$\frac{1}{\hat{\sigma}^2}\sum_{i=1}^{N}\varepsilon_i\hat{y}_i + \frac{\hat{\lambda}}{\hat{\sigma}}\sum_{i=1}^{N}\frac{f_i^*}{1-F_i^*}y_i = 0,$$

where f_i^* and F_i^* are standard normal density and distribution function, respectively, evaluated at $\varepsilon_i\hat{\lambda}\hat{\sigma}^{-1}$. Parameters of the one- and two-sided distributions are found by solving the identities

$$\sigma^2 = \sigma_u^2 + \sigma_v^2$$

$$\lambda = \frac{\sigma_u}{\sigma_v}.$$

Mean efficiency over the sample is given by

$$\exp\left(-\frac{\sqrt{2}}{\sqrt{\pi}}\right)\sigma_u,$$

and modal efficiency for each observation is given by

$$-\varepsilon(\sigma_u^2/\sigma^2).$$

Value

Returns a list with the following elements

`y.fit`	Estimated value of the frontier at X.fit
`gradient.fit`	Estimated gradient of the frontier at X.fit
`mean.efficiency`	Average efficiency for X, y as a whole
`mode.efficiency`	Modal efficiencies for each observation in X, y
`X.eval`	Matrix of inputs used for fitting
`X.constrained`	Matrix of inputs where constraints apply
`X.fit`	Matrix of inputs where curve is fit
`H.inv`	Inverse smoothing matrix used in fitting
`method`	Method used to fit frontier
`scaling.factor`	Factor by which constraints are multiplied before quadratic programming

References

Aigner D, Lovell CK, Schmidt P (1977). “Formulation and estimation of stochastic frontier production function models.” *Journal of econometrics*, **6**(1), 21–37.

Racine JS, Parmeter CF, Du P (2009). “Constrained nonparametric kernel regression: Estimation and inference.” Working paper.

Examples

```
data(USMacro)

USMacro <- USMacro[complete.cases(USMacro),]

#Extract data
X <- as.matrix(USMacro[,c("K", "L")])
y <- USMacro$Y

#Fit frontier
fit.sf <- fit.sf(X, y,
                 X.constrained = X,
                 method = "mc")

print(fit.sf$mean.efficiency)
# [1] 0.9772484

#Plot efficiency over time
library(ggplot2)

plot.df <- data.frame(Year = USMacro$Year,
                      Efficiency = fit.sf$mode.efficiency)

ggplot(plot.df, aes(Year, Efficiency)) +
  geom_line()
```

`H.inv.select` *Bandwidth matrix selection*

Description

Computes inverse of bandwidth matrix using rule-of-thumb from Silverman (1986).

Usage

```
H.inv.select(X, H.mult = 1)
```

Arguments

`X`	Matrix of inputs
`H.mult`	Scaling factor for rule-of-thumb smoothing matrix

Details

This method performs selection of (inverse) multivariate bandwidth matrices using Silverman's (1986) rule-of-thumb. Specifically, Silverman recommends setting the bandwidth matrix to

$$H_{jj}^{1/2} = \left(\frac{4}{M+2}\right)^{1/(M+4)} \times N^{-1/(M+4)} \times \text{sd}(x^j) \quad \text{for } j = 1, ..., M$$

$$H_{ab} = 0 \quad \text{for } a \neq b$$

where M is the number of inputs, N is the number of observations, and $\text{sd}(x^j)$ is the sample standard deviation of input j.

Value

Returns inverse bandwidth matrix

References

Silverman BW (1986). *Density estimation for statistics and data analysis*, volume 26. CRC press.

Examples

```
data(USMacro)

USMacro <- USMacro[complete.cases(USMacro),]

#Extract data
X <- as.matrix(USMacro[,c("K", "L")])

#Generate bandwidth matrix
print(H.inv.select(X))
#              [,1]         [,2]
# [1,] 3.642704e-08 0.000000e+00
# [2,] 0.000000e+00 1.215789e-08
```

`panel.production` *Randomly generated panel of production data*

Description

A dataset for illustrating technical and efficiency changes using smooth non-parametric frontiers.

Usage

```
panel.production
```

Format

A data frame with 200 observations of six variables.

Firm Firm identifier

Year Year of observation

X.1 Input 1

X.2 Input 2

X.3 Input 3

y Output

Details

Generated with the following code:

```
set.seed(100)

num.firms <- 20
num.inputs <- 3
num.years <- 10

beta <- runif(num.inputs, 0, 1)
TFP.trend = 0.25
TFP <- cumsum(rnorm(num.years)) + TFP.trend * (1:num.years)

sd.measurement <- 0.05
sd.inefficiency <- 0.01

f <- function(X){
  return(TFP + X
}
gen.firm.data <- function(i){
  X = matrix(runif(num.years * num.inputs, 1, 10), ncol = num.inputs)
  y = f(X) +
    rnorm(num.years, sd = sd.measurement) -
    abs(rnorm(num.years, sd = sd.inefficiency))
  firm.df <- data.frame(Firm = i,
                        Year = 1:num.years,
                        X = exp(X),
                        y = exp(y))
}

panel.production = Reduce(rbind, lapply(1:num.firms, gen.firm.data))
panel.production$Firm = as.factor(panel.production$Firm)
```

`reflect.data` *Data reflection for kernel smoothing*

Description

This function reflects data below minimum and above maximum for use in reducing endpoint bias in kernel smoothing.

Usage

```
reflect.data(X, y)
```

Arguments

`X`	Matrix of inputs
`y`	Vector of outputs

Value

Returns a list with the following elements

`X.reflected`	Reflected values of X
`y.reflected`	Reflected values of y

Examples

```
data(univariate)

#Extract data
X <- as.matrix(univariate$x)
y <- univariate$y

#Reflect data
reflected.data <- reflect.data(X, y)

X.reflected <- reflected.data$X
y.reflected <- reflected.data$y

#Plot
library(ggplot2)

plot.df <- data.frame(X = X.reflected,
                      y = y.reflected,
                      data = rep(c("reflected", "actual", "reflected"), each = nrow(X)))

ggplot(plot.df, aes(X, y)) +
  geom_point(aes(color = data))
```

---

technical.efficiency.change
Technical and efficiency change estimation

Description

Estimates technical and efficiency change using SNFA

Usage

```
technical.efficiency.change(df, input.var.names, output.var.name,
  firm.var.name, time.var.name, method = "u")
```

Arguments

`df`	Data frame with variables used in estimation
`input.var.names`	Names of input variables; must appear in df
`output.var.name`	Name of output variable; must appear in df
`firm.var.name`	Name of firm variable; must appear in df

`time.var.name` Name of time variable; must appear in df

`method` Constraints to apply; "u" for unconstrained, "m" for monotonically increasing, and "mc" for monotonically increasing and concave

Details

This function decomposes change in productivity into efficiency and technical change, as in Fare et al. (1994), using smooth non-parametric frontier analysis. Denoting $D_s(x_t, y_t)$ as the efficiency of the production plan in year t relative to the production frontier in year s, efficiency change for a given firm in year t is calculated as

$$\frac{D_{t+1}(x_{t+1}, y_{t+1})}{D_t(x_t, y_t)},$$

and technical change is given by

$$\left(\frac{D_t(x_{t+1}, y_{t+1})}{D_{t+1}(x_{t+1}, y_{t+1})} \times \frac{D_t(x_t, y_t)}{D_{t+1}(x_t, y_t)}\right)^{1/2}.$$

Value

Returns a data.frame with the following columns

`firm.var.name` Column of firm name data

`time.var.name` Column of time period data

`efficiency.change`
Average annual efficiency change since the previous period in data

`technical.change`
Average annual technical change since the previous period in data

`productivity.change`
Average annual productivity change since the previous period in data

References

Fare R, Grosskopf S, Norris M, Zhang Z (1994). "Productivity Growth, Technical Progress, and Efficiency Change in Industrialized Countries." *The American Economic Review*, **84**(1), 66-83.

Examples

```
data(panel.production)

results.df <- technical.efficiency.change(df = panel.production,
                                          input.var.names = c("X.1", "X.2", "X.3"),
                                          output.var.name = "y",
                                          firm.var.name = "Firm",
                                          time.var.name = "Year")

#Plot changes over time by firm
library(ggplot2)

ggplot(results.df, aes(Year, technical.change)) +
  geom_line(aes(color = Firm))
ggplot(results.df, aes(Year, efficiency.change)) +
```

```
  geom_line(aes(color = Firm))
ggplot(results.df, aes(Year, productivity.change)) +
  geom_line(aes(color = Firm))
```

`univariate` *Randomly generated univariate data*

Description

A dataset for illustrating univariate non-parametric boundary regressions and various constraints.

Usage

```
univariate
```

Format

A data frame with 50 observations of two variables.

x Input

y Output

Details

Generated with the following code:

```
set.seed(100)

N <- 50
x <- runif(N, 10, 100)
y <- sapply(x, function(x) 500 * x^0.25 - dnorm(x, mean = 70, sd = 10) * 8000) - abs(rnorm(N, sd = 20))
y <- y - min(y) + 10
df <- data.frame(x, y)
```

`USMacro` *US Macroeconomic Data*

Description

A dataset of real output, labor force, capital stock, wages, and interest rates for the U.S. between 1929 and 2014, as available. All nominal values converted to 2010 U.S. dollars using GDP price deflator.

Usage

```
USMacro
```

Format

A data frame with 89 observations of four variables.

Year Year

Y Real GDP, in billions of dollars

K Capital stock, in billions of dollars

K.price Annual cost of $1 billion of capital, using 10-year treasury

L Labor force, in thousands of people

L.price Annual wage for one thousand people

Source

https://fred.stlouisfed.org/

Index