

Package: DEA (via r-universe)

October 1, 2026

Version 1.0.3

Date 2026-09-28

Title Data Envelopment Analysis

Type Package

Maintainer David Bernstein <davebernstein1@gmail.com>

Description Nonparametric efficiency measurement by data envelopment analysis. Provides radial (Charnes-Cooper-Rhodes and Banker-Charnes-Cooper) technical efficiency under constant, variable, non-increasing and non-decreasing returns to scale, the slacks-based measure of Tone (2001), the additive model of Charnes and others (1985), and the directional distance function of Chambers, Chung and Fare (1996), all through one interface and one result object. Efficiency estimates are accompanied by peers, slacks, returns-to-scale classification, scale efficiency and the optimal multipliers, and by bias-corrected estimates and confidence intervals from the smoothed homogeneous bootstrap of Simar and Wilson (1998), which also supplies a test of constant returns to scale for the sample as a whole (Simar and Wilson, 2002). Where prices are known, cost, revenue and Nerlovian profit efficiency separate the technical component from the allocative one; where they are not, cross-efficiency with the secondary goals of Doyle and Green (1994) ranks units that a self-appraisal leaves tied. This package succeeds the archived 'DEA' package of Diaz-Martinez and Fernandez-Menendez (2008).

Suggests knitr, rmarkdown, testthat (>= 3.0.0), parallel, grDevices

Imports lpSolveAPI, stats, graphics, utils

Depends R (>= 4.0.0)

License GPL (>= 2)

Language en-US

URL <https://www.davidharrybernstein.com/software>,
<https://github.com/davidhbernstein/DEA>

BugReports <https://github.com/davidhbernstein/DEA/issues>

LazyLoad yes

LazyData true

NeedsCompilation no

VignetteBuilder knitr

Config/testthat/edition 3

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

Date/Publication 2026-10-01 16:08:49 UTC

RemoteUrl <https://github.com/davidhbernstein/dea>

RemoteRef HEAD

RemoteSha e2c20717abe3a05270469d7f0448ac44c4e0829a

Contents

DEA-package	3
charnes1981	5
dea	7
dea-methods	10
dea_add	12
dea_boot	15
dea_categorical	19
dea_cost	21
dea_cross	24
dea_ddf	26
dea_malmquist	29
dea_mult	31
dea_nd	35
dea_reg	37
dea_rts	40
dea_rts_test	42
dea_sbm	46
dea_sim	48
dea_subsample	51
dea_undesirable	54
dea_weak	57
dea_wr	60

Index	64
--------------	-----------

Description

Nonparametric efficiency measurement by data envelopment analysis. Provides radial (Charnes-Cooper-Rhodes and Banker-Charnes-Cooper) technical efficiency under constant, variable, non-increasing and non-decreasing returns to scale, the slacks-based measure of Tone (2001), the additive model of Charnes and others (1985), and the directional distance function of Chambers, Chung and Fare (1996), all through one interface and one result object. Efficiency estimates are accompanied by peers, slacks, returns-to-scale classification, scale efficiency and the optimal multipliers, and by bias-corrected estimates and confidence intervals from the smoothed homogeneous bootstrap of Simar and Wilson (1998), which also supplies a test of constant returns to scale for the sample as a whole (Simar and Wilson, 2002). Where prices are known, cost, revenue and Nerlovian profit efficiency separate the technical component from the allocative one; where they are not, cross-efficiency with the secondary goals of Doyle and Green (1994) ranks units that a self-appraisal leaves tied. This package succeeds the archived 'DEA' package of Diaz-Martinez and Fernandez-Menendez (2008).

Details

Technical efficiency – five estimators, one interface and one result object:

`dea` radial Debreu-Farrell efficiency under constant, variable, non-increasing or non-decreasing returns, or free disposal, in either orientation, with second-stage slacks and super-efficiency.

`dea_sbm` the slacks-based measure of Tone (2001).

`dea_ddf` the directional distance function of Chambers, Chung and Fare (1996), which handles zero and negative data.

`dea_add` the additive model of Charnes et al. (1985), unweighted or as the Range Adjusted Measure.

`dea_rts` scale efficiency and the returns-to-scale classification.

`dea_categorical`, `dea_nd` environments the DMU did not choose: an ordered category becomes a nested reference set, and a fixed input stops being something theta contracts.

`dea_mult` the multiplicative, or piecewise Cobb-Douglas, model – a log-linear envelope and a geometric peer target rather than a linear one and an arithmetic target. No other package on CRAN has it.

`dea_weak` the weak-disposability technology, where a bad output can be reduced only by scaling the activity back – the one route of the three that encodes null-jointness.

`dea_undesirable` efficiency when some outputs are bad, by the Seiford-Zhu transformation or by treating them as inputs – with the models the first of those is not valid for refused rather than served.

Over time, on a long-format panel:

`dea_panel` one row per DMU per period, with the period order taken from the data rather than guessed.

`dea_malmquist` the Malmquist productivity index of Fare, Grosskopf, Norris and Zhang (1994), split into efficiency change and technical change, and efficiency change into pure and scale parts.

When prices are known, efficiency splits into a technical part and a part that is about buying the wrong mix at those prices:

`dea_cost`, `dea_revenue` cost and revenue efficiency, each factoring exactly into technical times allocative.

`dea_profit` Nerlovian profit inefficiency, which adds rather than multiplies and ties back to the directional distance function.

When they are not, and the efficient set is too large to rank:

`dea_cross` cross-efficiency – peer appraisal rather than self-appraisal – with the Doyle-Green secondary goals, which bracket an answer that is otherwise not reproducible across solvers.

`multipliers` the optimal weights themselves, from the multiplier form of any radial fit.

`dea_wr`, `dea_cone` weight restrictions, assurance regions and the cone-ratio model, for when the freedom to choose any prices at all is the problem.

and, because a DEA score is an estimate and not a measurement,

`dea_boot` bias correction and confidence intervals by the smoothed homogeneous bootstrap of Simar and Wilson (1998).

`dea_subsample` the m out of n bootstrap, for when the full-sample one is not known to be consistent – and for free disposal, which `dea_boot()` refuses.

`dea_reg` the two-stage regression of Simar and Wilson (2007), for when efficiency is to be explained by covariates – the place applied DEA most often goes wrong.

`dea_rts_test` the test of constant returns for the sample as a whole, by the bootstrap of Simar and Wilson (2002), where `dea_rts()` only classifies.

`dea_rate` the rate at which any of this can converge, which depends on the number of inputs and outputs and is never root- n .

`dea_sim` a technology whose efficiency is known in closed form, for checking an estimator against a truth rather than against another implementation.

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

David Bernstein [aut, cre] (ORCID: <<https://orcid.org/0000-0002-2267-5741>>), Zuleyka Diaz-Martinez [aut], Jose Fernandez-Menendez [aut]

See Also

<https://www.davidharrybernstein.com/software>

<https://github.com/davidhbernstein/DEA>

Examples

```
sim <- dea_sim(100, p = 2, q = 1, seed = 1)

fit <- dea(sim$x, sim$y, rts = "vrs", orientation = "in")
summary(fit)

dea_sbm(sim$x, sim$y, rts = "vrs")
dea_ddf(sim$x, sim$y, direction = "both", rts = "vrs")
dea_rts(sim$x, sim$y)

## Productivity over time.
later <- dea_sim(100, p = 2, q = 1, seed = 2)
dea_malmquist(rbind(sim$x, later$x * 0.95), rbind(sim$y, later$y * 1.08),
  id = rep(1:100, 2), period = rep(1:2, each = 100))

## With prices: on the frontier, but at the wrong point on it?
dea_cost(sim$x, sim$y, w = c(1.2, 0.8), rts = "crs")

## Without prices: ranking the units that all scored 1.
dea_cross(sim$x, sim$y, secondary = "benevolent")
```

charnes1981

Program Follow Through: the original data envelopment analysis data set

Description

Seventy US primary school sites from Program Follow Through, a federally sponsored programme of remedial assistance to disadvantaged pupils. This is the data set on which Charnes, Cooper and Rhodes introduced and applied the model that carries their name, and it remains the standard worked example in the field.

Usage

```
data(charnes1981)
```

Format

A data frame with 70 rows and 10 columns.

site School site name. Not unique: 38 distinct names across 70 rows, since several cities contributed more than one site — New York and Philadelphia nine each.

pft Logical. TRUE for the 49 sites enrolled in Program Follow Through (rows 1–49), FALSE for the 21 that were not (rows 50–70).

x1 Input: education level of the mother.

x2 Input: highest occupation of a family member.

x3 Input: parental visits to the school.

x4 Input: time spent with children on school-related topics.

x5 Input: number of teachers at the site.

y1 Output: reading score.

y2 Output: mathematics score.

y3 Output: self-esteem score.

Details

Five inputs and three outputs on seventy units, which by the rule of thumb $n \geq 3(p + q)$ is comfortably enough — 70 against 24 — and is nonetheless a problem in eight dimensions. Expect a large efficient set, and see the discussion of the curse of dimensionality in [dea_rate](#) before reading much into the scores of the units that reach 1.

The pft flag makes this a natural example for an explicit reference technology: scoring the enrolled sites against the frontier spanned by the non-enrolled ones, or the reverse, is a metafrontier comparison and is what xref/yref are for.

Note

The site names are as distributed, including the misspellings “Chigago” and “Philidelphia”. They are left alone rather than silently corrected, so that this copy can be compared against the others without a diff appearing where none belongs.

Source

Charnes, A., Cooper, W. W. and Rhodes, E. (1981). Evaluating program and managerial efficiency: an application of data envelopment analysis to Program Follow Through. *Management Science* 27, 668–697.

The values here are identical to those distributed as `Benchmarking::charnes1981` and `npsf::ccr81`; the two were checked against each other before this copy was made and agree on every one of the 70×8 measurements. The columns have been renamed and the programme indicator turned into a logical, but no number has been altered.

References

Charnes, A., Cooper, W. W. and Rhodes, E. (1978). Measuring the efficiency of decision making units. *European Journal of Operational Research* 2, 429–444.

Examples

```
data(charnes1981)
x <- charnes1981[, c("x1", "x2", "x3", "x4", "x5")]
y <- charnes1981[, c("y1", "y2", "y3")]

## The model of the original paper: constant returns, input oriented.
ccr <- dea(x, y, rts = "crs", orientation = "in")
ccr

## Variable returns, and the scale decomposition between the two.
```

```
dea_rts(x, y, orientation = "in")

## Score the enrolled sites against the frontier spanned by the sites that
## were NOT enrolled -- a metafrontier comparison, not an internal ranking.
inp <- charnes1981$pft
dea(x[inp, ], y[inp, ], rts = "vrs", orientation = "in",
    xref = x[!inp, ], yref = y[!inp, ])

```

dea	<i>Radial technical efficiency</i>
-----	------------------------------------

Description

Debreu-Farrell radial efficiency by data envelopment analysis, under any of five returns-to-scale assumptions and either orientation, with optional second-stage slacks and Andersen-Petersen super-efficiency.

Usage

```
dea(x, y, data = NULL,
    rts = c("vrs", "crs", "nirs", "ndrs", "fdh"),
    orientation = c("in", "out"),
    slack = TRUE, super = FALSE, multipliers = FALSE,
    peers = TRUE, scaling = TRUE,
    xref = NULL, yref = NULL, dataref = NULL)

```

Arguments

x	Inputs, one row per decision-making unit (DMU) and one column per input. A matrix, a data frame, or a numeric vector for a single input. With data supplied it may instead be a character vector of column names or a one-sided formula such as <code>~ labour + capital</code> .
y	Outputs, in the same forms as x.
data	Optional data frame holding the columns named by x and y.
rts	Returns to scale, i.e. the restriction placed on the intensity weights: "vrs" variable, $\sum \lambda = 1$; the BCC model of Banker, Charnes and Cooper (1984). The default. "crs" constant, λ unrestricted; the CCR model of Charnes, Cooper and Rhodes (1978). "nirs" non-increasing, $\sum \lambda \leq 1$. "ndrs" non-decreasing, $\sum \lambda \geq 1$. "fdh" free disposal hull, λ binary; drops convexity altogether (Deprins, Simar and Tulkens 1984).

	Matching ignores case, and "drs"/"irs" are accepted as aliases for "nirs"/"ndrs", which is how Benchmarking spells them.
orientation	"in" contracts inputs at fixed output and returns $\theta \in (0, 1]$; "out" expands outputs at fixed input and returns $\phi \geq 1$. Both equal 1 on the frontier. Use <code>efficiency(fit, "score")</code> to put either on a common $(0, 1]$ scale.
slack	Run the second-stage slack-maximizing program. Its cost is small – the constraint matrix does not depend on the DMU – and without it the efficient component reports radial efficiency rather than Pareto-Koopmans efficiency. See 'Details'.
super	Andersen-Petersen (1993) super-efficiency: score each DMU against a technology it is excluded from, which breaks the ties among the efficient units. See 'Details' for the infeasibility that comes with it.
multipliers	Also solve the <i>multiplier</i> (dual) program at each DMU and return the optimal weights v on the inputs, u on the outputs and the returns-to-scale intercept u_0 . Off by default because it is a second linear program per DMU and most callers want the score, which strong duality makes identical either way. Not available for <code>rts = "fdh"</code> , which is not convex and so has no multiplier form. See 'Details' and <code>multipliers</code> .
peers	Keep the $n \times n$ matrix of intensity weights. Set FALSE for large samples, where it is the dominant memory cost.
scaling	Rescale every input and output column to mean one before solving. Radial efficiency is invariant to the units of each variable, so this changes no answer and conditions the linear program considerably better; slacks are returned in the original units either way.
xref, yref	An explicit reference technology: the DMUs that SPAN the frontier, where x and y are the DMUs SCORED against it. Both must be given together. Default NULL uses x and y themselves, which is ordinary DEA.
dataref	Data frame for <code>xref/yref</code> when those are given as names or formulas. Defaults to data.

Details

The radial score is not the whole story. A DMU projected onto a vertical or horizontal face of the frontier is radially efficient and still dominated: some input can be cut further without giving up any output. The second stage maximizes total slack at the radial projection, and `efficient` combines the two into the Pareto-Koopmans judgement. Where the two differ, `print()` says so. `dea_sbm` builds the same judgement into a single number.

Peer sets are not unique. These linear programs frequently have multiple optima, so `lambda` is one valid set of benchmarks rather than the set. Efficiency scores are unique; the weights that achieve them generally are not, and no DEA package can make them so.

Super-efficiency and infeasibility. Removing a DMU from its own reference set can leave the program with no solution at all: under "vrs", "nirs" or "ndrs" a DMU at the edge of the input space may have no other unit able to dominate it. `eff` is NA there. That is the correct answer and not a numerical failure – a package that returns a large finite number instead is reporting something the model does not define. Infeasibility cannot occur for input-oriented "crs".

The curse of dimensionality. DEA is consistent but converges slowly, at a rate that worsens with every variable added: see [dea_rate](#). With $n < 3(p + q)$ most DMUs are efficient by dimension alone and `dea()` warns.

Sampling variation. Because the frontier is spanned by the observed DMUs it lies inside the true one, so every score is biased toward 1. See [dea_boot](#) for the bias correction and confidence intervals.

Value

An object of class "dea": a list whose components are

<code>eff</code>	Named numeric vector of efficiency scores, on the orientation's own scale.
<code>model, rts, orientation, super</code>	The specification, as resolved.
<code>lambda</code>	Matrix of intensity weights, evaluated DMUs by reference DMUs, or NULL when <code>peers = FALSE</code> .
<code>sum_lambda</code>	Row sums of <code>lambda</code> from the first-stage program.
<code>slack_x, slack_y</code>	Input and output slacks in the original units, or NULL when <code>slack = FALSE</code> .
<code>efficient</code>	Logical. Pareto-Koopmans efficiency when slacks were computed – on the frontier radially AND with no slack left – and radial efficiency otherwise.
<code>status</code>	Per-DMU solver code: 0 an optimal solve, 2 an infeasible program, anything else a numerical failure. The three call for different responses – nothing, a statement about the data, and a conditioning problem – and DMUs that are not solved are reported as NA with a warning saying which of the three it was. Under <code>rts = "fdh"</code> no linear program is solved, but the codes mean the same thing: a DMU with no dominating peer is reported as 2, so the field can be read the same way across every technology.
<code>mult_status</code>	The same, for the multiplier (dual) program, and present only when <code>multipliers = TRUE</code> . It is a <i>separate</i> field because it can differ from <code>status</code> : the two are different linear programs, and a DMU whose score solved can still have weights that did not. Failures are reported with their own warning. The codes do not mean quite what they mean for the envelopment program. If that program is feasible and bounded then strong duality makes its dual feasible and bounded too, so an infeasible dual is a contradiction rather than a statement about the data — which is why the dual is retried on a fresh program where the envelopment sweep would not bother. The converse case is ordinary: under <code>super = TRUE</code> an infeasible envelopment program has an unbounded dual, and both are reported, one as NA score and the other as NA weights.
<code>x, y, xref, yref</code>	The data, as coerced.
<code>n, nref, p, q</code>	Counts of evaluated DMUs, reference DMUs, inputs and outputs.
<code>total_time</code>	Elapsed seconds.
<code>call</code>	The matched call.

References

- Andersen, P. and Petersen, N. C. (1993). A procedure for ranking efficient units in data envelopment analysis. *Management Science* 39, 1261–1264.
- Banker, R. D., Charnes, A. and Cooper, W. W. (1984). Some models for estimating technical and scale inefficiencies in data envelopment analysis. *Management Science* 30, 1078–1092.
- Charnes, A., Cooper, W. W. and Rhodes, E. (1978). Measuring the efficiency of decision making units. *European Journal of Operational Research* 2, 429–444.
- Deprins, D., Simar, L. and Tulkens, H. (1984). Measuring labor-efficiency in post offices. In M. Marchand, P. Pestieau and H. Tulkens (eds), *The Performance of Public Enterprises*. North-Holland.

See Also

[dea_sbm](#), [dea_ddf](#), [dea_rts](#), [dea_boot](#), [efficiency](#), [dea_sim](#)

Examples

```
sim <- dea_sim(60, p = 2, q = 1, seed = 1)

fit <- dea(sim$x, sim$y, rts = "vrs", orientation = "in")
fit
head(efficiency(fit))
head(peers(fit))

## The same call from a data frame, naming columns.
d <- sim$data
dea(x = c("x1", "x2"), y = "y1", data = d, rts = "crs")

## Score one group against another group's frontier.
dea(sim$x[1:10, ], sim$y[1:10, , drop = FALSE], rts = "vrs",
    xref = sim$x[11:60, ], yref = sim$y[11:60, , drop = FALSE])
```

dea-methods

Methods for objects of class "dea"

Description

Extractors and display methods shared by [dea](#), [dea_sbm](#) and [dea_ddf](#), which all return the same class.

Usage

```
efficiency(object, ...)
## S3 method for class 'dea'
efficiency(object, type = c("natural", "score"), ...)

peers(object, ...)
```

```

## S3 method for class 'dea'
peers(object, threshold = 1e-08, ...)

slacks(object, ...)
## S3 method for class 'dea'
slacks(object, ...)

multipliers(object, ...)
## S3 method for class 'dea'
multipliers(object, ...)
## S3 method for class 'dea_cross'
multipliers(object, ...)

## S3 method for class 'dea'
print(x, ...)
## S3 method for class 'dea'
summary(object, ...)
## S3 method for class 'dea'
plot(x, ngrid = 400, ...)
## S3 method for class 'dea'
fitted(object, ...)
## S3 method for class 'dea'
nobs(object, ...)

```

Arguments

object, x	An object of class "dea", or – for <code>multipliers()</code> – one of class "dea_cross".
type	"natural" returns the model's own measure — θ , ϕ , ρ or β . "score" maps all of them onto $(0, 1]$ with 1 meaning efficient, which is what a table comparing models needs.
threshold	Intensity weights at or below this are not reported as peers.
ngrid	Number of grid points used to trace the frontier in <code>plot()</code> .
...	Passed to the underlying plot, or ignored.

Details

`efficiency(object, "score")` deliberately *fails* for a directional fit whose direction is neither purely input nor purely output. β is an additive distance measured in units of g and there is no ratio it corresponds to; a formula such as $(1 - \beta)/(1 + \beta)$ would produce a number that looks comparable to a Farrell score and is not.

`fitted()` returns the frontier projection: where each DMU would sit if it were efficient. For a radial fit with slacks that is the radial projection with the remaining slack removed, so the result is a Pareto-Koopmans point.

`plot()` draws the estimated frontier when there is one input and one output, and the distribution of scores otherwise. The frontier is traced by *evaluating* the fitted technology on a grid of input levels rather than by joining up the efficient DMUs, so the same code draws it correctly under every

returns-to-scale assumption — including the free disposal hull's staircase — and puts the vertical and horizontal faces where they actually are.

Peer sets are not unique. These linear programs frequently have multiple optima, so `peers()` reports one valid benchmark set rather than the benchmark set. Efficiency scores are unique; the weights achieving them generally are not.

Value

`efficiency` and `nobs` return numeric vectors; `peers` returns a data frame with columns `dmu`, `peer` and `lambda`; `slacks` and `fitted` return matrices with one row per DMU. `print` and `summary` return their argument invisibly.

The multiplier form

`multipliers()` returns one row per DMU: the weights v on the inputs, u on the outputs, and u_0 where the technology has one. They are the prices that make the DMU look as good as it possibly can while valuing no reference DMU above break-even, and they satisfy $v'x_o = 1$ (input orientation) in the caller's own units.

For an efficient DMU they are not unique. An efficient unit generally has a whole face of optimal weight vectors, every one giving it the same score of 1 and giving other DMUs different ones, so what comes back is one vertex of that face and another solver may return a different vertex. The score is unique; the prices supporting it are not. [dea_cross](#) exists largely to deal with the consequences.

See Also

[dea](#), [dea_sbm](#), [dea_ddf](#)

Examples

```
sim <- dea_sim(60, p = 1, q = 1, seed = 1)
fit <- dea(sim$x, sim$y, rts = "vrs")

summary(fit)
head(efficiency(fit, "score"))
head(peers(fit))
head(slacks(fit))
head(fitted(fit))

plot(fit)
```

dea_add

Additive efficiency, and the weighted measures built on it

Description

The additive model of Charnes, Cooper, Golany, Seiford and Stutz (1985): maximize total slack over the technology, with no orientation and no radial contraction. Available unweighted, or normalized as the Range Adjusted Measure or the Measure of Inefficiency Proportions.

Usage

```
dea_add(x, y, data = NULL,
        measure = c("ram", "mip", "unweighted"),
        rts = c("vrs", "crs", "nirs", "ndrs"),
        peers = TRUE, scaling = TRUE,
        xref = NULL, yref = NULL, dataref = NULL)
```

Arguments

- | | |
|-------------------------------------|--|
| x, y, data | Inputs and outputs, in any of the forms accepted by dea . <code>measure = "mip"</code> additionally requires every value to be strictly positive, because it divides each slack by the DMU's own level. |
| measure | Which weights the objective carries:
<div style="margin-left: 20px;"> <p>"ram" Range Adjusted Measure (Cooper, Park and Pastor 1999). Each slack is divided by the range of its variable across the reference set, and by $p + q$. Units invariant, and the reported score $\rho = 1 - \text{objective}$ lies in $[0, 1]$, because no slack can exceed its variable's range. The default.</p> <p>"mip" Measure of Inefficiency Proportions. Each slack is divided by the evaluated DMU's own level of that variable, and by $p + q$. Units invariant, and at most 1 — but not bounded below by 0: an input slack cannot exceed the DMU's own input, so its term is capped, while an output slack has no such ceiling and a badly performing DMU can score negative. RAM is bounded on both sides, which is what it was designed for, and is the better default.</p> <p>"unweighted" The original 1985 model. Reported as the raw slack total, so 0 is efficient and larger is worse — and it carries the units of the data. See 'Details'.</p> </div> |
| rts | Returns to scale, as in dea . There is no "fdh" option — the measure is defined on a convex technology. |
| peers, scaling, xref, yref, dataref | As in dea , except that scaling is ignored for <code>measure = "unweighted"</code> : that measure is defined on the caller's units, so rescaling the columns would change the answer rather than leave it alone. |

Details

One model, three scales. All three options solve the same program over the same technology and differ only in the objective weights. The literature often presents RAM and MIP as separate models; they are one model read three ways, and the projection and the efficient set are identical across them.

The classic version is not units invariant, and that is why the other two exist. With unit weights the objective adds a slack measured in staff-hours to one measured in euros: double an input's units and its slack halves, which can change which DMU looks worst. The efficient *set* is unaffected — a slack is zero in any units — but the *ordering* of the inefficient DMUs is not. `efficiency(fit, "score")` therefore refuses to put the unweighted measure on a common scale, and `scaling = TRUE` is ignored for it.

Solving in the caller's units also means the unweighted program is poorly conditioned when the columns differ wildly in magnitude, and `dea_add()` warns above a spread of 10^3 . On a design with

one input multiplied by 1000, one DMU failed numerically outright. That is a real property of the measure and another reason to prefer "ram".

No orientation. The additive model does not ask by what factor inputs could shrink or outputs grow; it asks how far the DMU is from the frontier in every coordinate at once. That makes it the natural tool when the question is *which* units are dominated rather than *by how much*, and it is why its efficient set coincides exactly with [dea_sbm](#)'s $\rho = 1$ set and with [dea](#)'s Pareto-Koopmans set.

Infeasibility with an external reference set. As for [dea_sbm](#), the balance rows are equalities with non-negative slacks, so a solution exists only if some convex combination of the reference DMUs weakly dominates the evaluated point in every input and output. A point outside the estimated technology has none; `dea_add()` warns and reports NA. For the same reason "ram" can return a score below 0 when scoring out of sample, since a slack may then exceed the reference range.

Value

An object of class "dea" with `model = "additive"` and the chosen measure recorded. `efficient` is the Pareto-Koopmans efficient set on every scale.

References

Charnes, A., Cooper, W. W., Golany, B., Seiford, L. and Stutz, J. (1985). Foundations of data envelopment analysis for Pareto-Koopmans efficient empirical production functions. *Journal of Econometrics* 30, 91–107.

Cooper, W. W., Park, K. S. and Pastor, J. T. (1999). RAM: a range adjusted measure of inefficiency for use with additive models, and relations to other models and measures in DEA. *Journal of Productivity Analysis* 11, 5–42.

See Also

[dea_sbm](#), [dea](#), [dea_ddf](#)

Examples

```
data(charnes1981)
x <- charnes1981[, paste0("x", 1:5)]
y <- charnes1981[, paste0("y", 1:3)]

dea_add(x, y, measure = "ram", rts = "vrs")

## The efficient set is the Pareto-Koopmans set, identical to the one the
## radial model finds after its second stage and the one dea_sbm() scores 1.
a <- dea_add(x, y, measure = "ram", rts = "vrs")
r <- dea(x, y, rts = "vrs", orientation = "in", slack = TRUE)
identical(unname(a$efficient), unname(r$efficient))

## RAM is units invariant; the unweighted model is not.
x2 <- x; x2[, 1] <- x2[, 1] * 1000
max(abs(dea_add(x2, y, measure = "ram")$eff - dea_add(x, y, measure = "ram")$eff))
max(abs(dea_add(x2, y, measure = "unweighted")$eff -
  dea_add(x, y, measure = "unweighted")$eff))
```

dea_boot	<i>Bias correction and confidence intervals by the Simar-Wilson bootstrap</i>
----------	---

Description

The smoothed homogeneous bootstrap of Simar and Wilson (1998): bias-corrected efficiency estimates and confidence intervals for a radial DEA fit.

Usage

```
dea_boot(object, B = 2000, alpha = 0.05,
         bw = c("silverman", "nrd0", "ucv", "sj"),
         seed = NULL, ncores = 1L, progress = interactive())

## S3 method for class 'dea_boot'
print(x, ...)
## S3 method for class 'dea_boot'
summary(object, ...)
## S3 method for class 'dea_boot'
plot(x, sort = TRUE, ...)
```

Arguments

object	A fit from dea , scored against its own sample (no xref), not a super-efficiency or free-disposal-hull fit. For the print method, an object of class "dea_boot".
B	Bootstrap replications. 2000 is the usual recommendation; each one costs a full DEA sweep. The run is not quite B times the cost of the fit, because the original fit and the reference technology are computed once and do not scale with B: measured, that fixed part is about half the cost of a B = 200 run, so ten times the replications costs about five times the time rather than ten.
alpha	One minus the nominal coverage of the interval.
bw	Bandwidth for the kernel smoothing, as a rule name or a positive number. "silverman" is the robust normal-reference rule Simar and Wilson use, applied to the reflected sample; the others are bw.nrd0 , bw.ucv and bw.SJ .
seed	Integer seed. The caller's random-number stream is restored afterwards.
ncores	Replications run in parallel through <code>parallel::mclapply</code> when this exceeds 1. Not available on Windows; a request there runs serially with a message.
progress	Print a counter while running serially.
x	An object of class "dea_boot".
sort	For plot, order the DMUs by their bias-corrected estimate. FALSE keeps the data order.
...	Ignored.

Details

Why it is needed. The DEA frontier is spanned by the observed DMUs, so it lies inside the true one: every efficiency score is biased toward 1, and the bias is not small. A DEA score reported on its own is a point estimate of unknown accuracy from an estimator with a known, one-signed bias. This is the largest gap between how DEA is used in practice and what is known about it.

Smoothing and reflection. Resampling the estimated efficiencies directly is inconsistent: it puts an atom of probability on each observed score, in particular a large one on 1, so the pseudo-frontier is a fixed set of points rather than a draw from a density. Smoothing with a Gaussian kernel fixes that but would leak probability past the boundary at 1, exactly where the mass is. The remedy is reflection: smooth the $2n$ -point set $\{\hat{\theta}_i\} \cup \{2 - \hat{\theta}_i\}$, which is symmetric about 1 by construction, then fold draws back across it.

Whether to use the correction. `bias_corrected` is $2\hat{\theta} - \text{mean}(\hat{\theta}^*)$. Simar and Wilson warn against applying it unconditionally: it removes a bias but adds the variance of the estimate of that bias, and is a net loss unless the bias is large relative to the bootstrap standard error. Their rule of thumb — correct only when $|bias|/se > 1/\sqrt{3}$ — is evaluated per DMU and reported as `correct_worthwhile`, rather than being applied silently.

Intervals. The interval is the percentile interval for $\hat{\theta} - \theta$, inverted, and so is centred near `bias_corrected` rather than near `eff`.

Value

An object of class "dea_boot". Its table component is a data frame with one row per DMU and columns `dmu`, `eff`, `bias`, `se`, `bias_corrected`, `ci_lower`, `ci_upper` and `correct_worthwhile`. The full n by B matrix of bootstrap scores is returned as `boot`, and the bandwidth actually used as `bw`.

The algorithm, step by step

Each step of Algorithm #1 of Simar and Wilson (1998), and what this function does for it. Stated explicitly because the details are where bootstrap procedures for frontier models differ from one another, and two implementations that describe themselves the same way can be doing different things.

1. *Fit.* $\hat{\theta}_i$ for $i = 1, \dots, n$ comes from the supplied `dea` object; it is not recomputed here. The fit must be scored against its own sample, because the procedure resamples the same units that span the technology.
2. *Reflect.* Form the $2n$ -point set $\{\hat{\theta}_i\} \cup \{2 - \hat{\theta}_i\}$. This is exact in either orientation: $\theta \in (0, 1]$ and $\phi \in [1, \infty)$ both have their boundary at 1.
3. *Bandwidth.* h is computed on the reflected set of $2n$ points, not on the n original scores. The default "silverman" is the robust normal-reference rule $0.9 \min(s, IQR/1.349)(2n)^{-1/5}$. The alternatives are offered because the choice is genuinely open — the reflected density is bimodal enough that a normal reference oversmooths it — and a positive number may be given directly.
4. *Resample and smooth.* Draw β_i^* with replacement from the reflected set and set $\tilde{\theta}_i^* = \bar{\beta}^* + (\beta_i^* + h\varepsilon_i - \bar{\beta}^*)/\sqrt{1 + h^2/s_{refl}^2}$, with ε_i standard normal. The denominator is the variance correction: adding $h\varepsilon$ inflates the variance by h^2 , and without rescaling the pseudo-sample is overdispersed.

5. *Fold.* Map draws back across the boundary — $\theta^* = 2 - \tilde{\theta}^*$ where $\tilde{\theta}^* > 1$ for an input orientation, and the mirror for an output orientation. The smoothed density is symmetric about 1 by construction, so folding is measure preserving and no mass crosses the boundary.
6. *Build the pseudo-sample.* Move each DMU along its own orientation so that its efficiency *relative to the estimated frontier* becomes θ_i^* instead of $\hat{\theta}_i$: $x_i^* = x_i \hat{\theta}_i / \theta_i^*$ for an input orientation, $y_i^* = y_i \hat{\theta}_i / \theta_i^*$ for an output one.
7. *Re-estimate.* The **technology** is the pseudo-sample; the points **evaluated** against it are the original observations. Scoring the pseudo-points against themselves would simply reproduce $\hat{\theta}$.
8. *Read off.* Over B replications: $bias = \text{mean}(\hat{\theta}^*) - \hat{\theta}$, $se = \text{sd}(\hat{\theta}^*)$, and $bias_corrected = \hat{\theta} - bias = 2\hat{\theta} - \text{mean}(\hat{\theta}^*)$. The interval is the percentile interval on $\hat{\theta}^* - \hat{\theta}$ inverted, i.e. $2\hat{\theta} - q_{1-\alpha/2}$ to $2\hat{\theta} - q_{\alpha/2}$, which is an interval for θ rather than for $\hat{\theta}$.

Bootstrap scores routinely cross 1 — θ^* above it, ϕ^* below — because the pseudo-technology is a shrunk copy of the estimated one and an original observation need not lie inside it. That tail is the interval, and it is not clamped.

What is assumed, and when it fails

Homogeneity. This is the *homogeneous* bootstrap: it assumes the distribution of inefficiency does not depend on the input-output mix, so that one common density of θ may be resampled for every DMU. Where inefficiency varies systematically with the inputs — large units systematically closer to the frontier than small ones, say — the assumption fails and the procedure is not consistent. The remedy is the heterogeneous bootstrap of Simar and Wilson (2000), which conditions the resampled density on (x, y) and is **not implemented here**.

Checking homogeneity is harder than it looks. The obvious check is to regress the estimated scores on the inputs and outputs and read off an F test. Do not: it rejects homogeneity on data that satisfy it, essentially always. Across 200 samples from `dea_sim`, which draws inefficiency independently of x by construction, that test rejected at the nominal 10, 5 and 1 per cent levels in 100 per cent of samples. The reason is that the estimator manufactures the association being looked for — the scores are not independent of one another, they are all measured against one frontier estimated from the whole sample, and the DEA bias is larger in sparse regions of the input space than in dense ones. On the samples above the correlation between x and the *true* θ is +0.04 and the correlation between x and the *estimated* $\hat{\theta}$ is −0.36. This is the same dependence that invalidates conventional standard errors in two-stage DEA regression (Simar and Wilson, 2007).

Replacing the F distribution with a null simulated from the fitted homogeneous model removes that objection but does not produce a usable test either; the measurement is in `horserace/homogeneity_experiment.R` of the package's development repository. So the assumption is documented here rather than tested, and a user who suspects it fails should treat the intervals with corresponding caution rather than look for a diagnostic in this package.

Convexity. The resampling scheme is built for the convex DEA estimator. Under `rts = "fdh"` the relevant procedure is the subsampling bootstrap of Jeong and Simar (2006), not this one, and `dea_boot` refuses a free-disposal-hull fit rather than returning something inapplicable.

Super-efficiency. Refused. The Andersen-Petersen score is not an estimator of a distance to the true frontier, so there is no bias to correct.

Boundary DMUs. The sampling distribution of a DMU estimated as efficient is one-sided, and intervals are hardest exactly there. Coverage should be judged separately for frontier and interior

units rather than averaged; the harness in horserace/ of the package's development repository reports the split for that reason, and the split is where this implementation's calibration actually lies. On a variable-returns design with two inputs and one output, 300 replications at $B = 200$, these intervals cover 0.887 of the frontier units and 0.593 of the interior ones at $n = 50$, and 0.829 against 0.723 at $n = 200$, against a nominal 0.95. The frontier coverage is the best of the four implementations raced; the interior coverage is not. Read a single averaged coverage number for this or any other implementation with that in mind.

Choosing B. 2000 is the usual recommendation. Interval endpoints are sample quantiles, so a small B interpolates the tails from few order statistics, and going from 200 to 2000 is worth about +0.008 in coverage here – real, measured over 300 replications on paired data, and small next to how far every implementation of this procedure sits below nominal. Cost is affine rather than linear in B: the original fit and the reference technology are computed once, and on the measured design they account for about half of a $B = 200$ run, so raising B tenfold costs about five times the time.

References

- Simar, L. and Wilson, P. W. (1998). Sensitivity analysis of efficiency scores: how to bootstrap in nonparametric frontier models. *Management Science* 44, 49–61.
- Simar, L. and Wilson, P. W. (2000). A general methodology for bootstrapping in nonparametric frontier models. *Journal of Applied Statistics* 27, 779–802.
- Simar, L. and Wilson, P. W. (2007). Estimation and inference in two-stage, semi-parametric models of production processes. *Journal of Econometrics* 136, 31–64.
- Jeong, S.-O. and Simar, L. (2006). Linearly interpolated FDH efficiency score for nonconvex frontiers. *Journal of Multivariate Analysis* 97, 2141–2161.

See Also

[dea_subsample](#) for the m out of n alternative, which is what free disposal needs and what Simar and Wilson (2011) recommend when the full-sample bootstrap's consistency is in doubt; [dea](#), [dea_rate](#)

Examples

```
sim <- dea_sim(40, p = 1, q = 1, seed = 3)
fit <- dea(sim$x, sim$y, rts = "vrs", orientation = "in")

## B is tiny here so the example runs quickly; use 2000 in practice.
b <- dea_boot(fit, B = 50, seed = 1, progress = FALSE)
b
head(b$table)

## The correction moves the estimates toward the truth.
c(raw = mean(fit$eff - sim$theta),
  corrected = mean(b$table$bias_corrected - sim$theta))
```

dea_categorical	<i>Categorical environmental variables, as nested reference sets</i>
-----------------	--

Description

Score each DMU only against units whose environment was no more favourable than its own. The categorical model of Banker and Morey (1986b), implemented by restricting the reference set rather than by their mixed-integer program.

Usage

```
dea_categorical(x, y, category, data = NULL,
               rts = c("vrs", "crs", "nirs", "ndrs"),
               orientation = c("in", "out"),
               slack = TRUE, peers = TRUE, scaling = TRUE)

## S3 method for class 'dea_categorical'
print(x, ...)
## S3 method for class 'dea_categorical'
summary(object, ...)
```

Arguments

<code>x, y, data</code>	Inputs and outputs, in any of the forms accepted by dea . For the print method, an object of class "dea_categorical".
<code>category</code>	The environment, as an ordered factor or a numeric rank, one entry per DMU, or a column name when data is given. Higher means more favourable . An unordered factor is refused.
<code>rts, orientation, slack, peers, scaling</code>	As in dea .
<code>object</code>	An object of class "dea_categorical".
<code>...</code>	Ignored.

Details

A school in a deprived catchment, a hospital in a rural district, a branch in a small town: the DMU operates in an environment it did not choose and cannot change, and scoring it against units in easier environments measures the environment rather than the management. The environment is *ordered* but is not a quantity — "more deprived" is a rank, not a number of units of anything — so it cannot simply be added as an input.

The fix is a restriction on who may be a peer, and nothing else. Order the categories from least to most favourable; a DMU in category k is scored against every DMU in categories $1..k$. That is xref/yref applied once per category: no new program and no integer variables. Two identities follow and are worth knowing, because they make the output readable: the *top* category's scores are exactly the pooled ones, and the *bottom* category's are exactly an ordinary fit on that subset alone.

Banker and Morey's own formulation is deliberately not built. Theirs is a mixed-integer program, and Charnes, Cooper, Lewin and Seiford (1994, ch. 3) are explicit that it "incorrectly prescribes categories that are undefined or meaningless" — it can hand a DMU a target in a category that does not exist. Adapting the solution procedure rather than the model is both correct and allows several categorical variables at once, which the mixed-integer version does not.

Which end is which is asked for explicitly, because getting it backwards does not error, it silently answers the opposite question. category must be an ordered factor or a numeric rank; an unordered factor is refused, since alphabetical order is not an ordering of environments. The reference-set size per category is printed so the direction is visible in the output rather than taken on trust.

Nesting costs sample size, and the cost falls on category 1, which is scored against its own members alone. A design that is comfortable pooled can be far below the $n \geq 3(p + q)$ rule of thumb once split, with almost everything efficient by dimension rather than by performance. The scores stay finite and plausible, so a warning is the only notice there is, and it is raised per category with the category named.

If the categories are not nested, this is the wrong function. Public versus private universities are not ordered — neither environment's advantages are a subset of the other's — and inventing an order produces a number that answers nothing. The right analysis there is a separate one per group, or a metafrontier through xref/yref. summary() says so.

Value

An object of class c("dea_categorical", "dea"). Beyond the usual components:

category, category_levels, n_category

The environment as an ordered factor, its levels from least to most favourable, and how many there are.

reference_size How many DMUs each unit was scored against.

pooled_eff What the same fit gives with no categories at all.

category_effect

How much the DMU's efficiency improves, on the common $(0, 1]$ scale, from being compared only with units no better placed. Never negative.

References

Banker, R. D. and Morey, R. C. (1986b). The use of categorical variables in data envelopment analysis. *Management Science* 32, 1613–1627.

Charnes, A., Cooper, W. W., Lewin, A. Y. and Seiford, L. M. (1994). *Data Envelopment Analysis: Theory, Methodology and Application*. Kluwer.

See Also

[dea_nd](#) for variables that are continuous but outside the DMU's control, [dea](#).

Examples

```
set.seed(21)
n <- 90
x <- matrix(runif(n * 2, 2, 10), n, 2,
```

```

      dimnames = list(NULL, c("staff", "budget")))
env <- rep(1:3, length.out = n)
y <- matrix((x[, 1] * x[, 2])^0.4 * c(1, 1.15, 1.32)[env] *
  exp(-abs(rnorm(n, 0, 0.2))), n, 1, dimnames = list(NULL, "output"))
catg <- factor(c("deprived", "average", "affluent")[env],
  levels = c("deprived", "average", "affluent"), ordered = TRUE)

r <- dea_categorical(x, y, catg)
r

## The top category is scored against everyone, so it gains nothing.
tapply(r$category_effect, r$category, mean)

```

dea_cost	<i>Cost, revenue and profit efficiency</i>
----------	--

Description

Efficiency when prices are known. Technical efficiency asks whether a DMU is on the frontier; these ask whether it is at the right *point* on it. A DMU can be perfectly efficient technically and still be buying the wrong input mix for the prices it faces, and only a price-based measure sees that.

Usage

```

dea_cost(x, y, w, data = NULL,
  rts = c("vrs", "crs", "nirs", "ndrs"),
  peers = TRUE, scaling = TRUE,
  xref = NULL, yref = NULL, dataref = NULL)

dea_revenue(x, y, r, data = NULL,
  rts = c("vrs", "crs", "nirs", "ndrs"),
  peers = TRUE, scaling = TRUE,
  xref = NULL, yref = NULL, dataref = NULL)

dea_profit(x, y, w, r, data = NULL,
  direction = c("both", "in", "out", "unit", "mean", "range"),
  rts = c("vrs", "nirs"),
  peers = TRUE, scaling = TRUE,
  xref = NULL, yref = NULL, dataref = NULL)

## S3 method for class 'dea_price'
plot(x, ...)

```

Arguments

x, y, data Inputs and outputs, in any of the forms accepted by [dea](#).

w	Input prices. Either a length- p numeric vector, meaning one price list faced by every DMU, or an $n \times p$ matrix or data frame with one row per DMU, or – with data – column names or a one-sided formula. All prices must be strictly positive. A bare vector is read as a shared price list, <i>not</i> as one observation per DMU; when $n = p$ makes the two readings indistinguishable, the call is refused rather than guessed.
r	Output prices, in the same forms, with q in place of p .
rts	Returns to scale, as in dea . <code>dea_profit()</code> accepts only "vrs" and "nirs"; see Details .
direction	For <code>dea_profit()</code> , the direction g the profit gap is normalized by, in the forms dea_ddf accepts — including "range", which normalizes by the value of the distance to the ideal point. The one direction it cannot use is a zero one: the gap is divided by $w'g_x + r'g_y$, so a DMU at the ideal point, whose range direction is zero by construction, is an error here where dea_ddf answers it with $\beta = 0$.
peers, scaling, xref, yref, dataref	As in dea .
...	Passed to the underlying plotting call.

Details

The decomposition is the point. Cost efficiency is

$$CE_o = \frac{\min\{w'_o x : (x, y_o) \in T\}}{w'_o x_o} = \theta_o \times AE_o$$

the first factor being the input-oriented radial score and the second the loss from using the wrong mix at these prices. Both are in $(0, 1]$, and a DMU is cost efficient only if it is both technically and allocatively efficient. Revenue efficiency decomposes the same way against the output-oriented score.

Profit is not a ratio, and this implementation does not pretend it is. Observed profit is routinely zero and sometimes negative, so Π_o/Π^* is undefined or meaningless exactly where the question matters. The measure used is the Nerlovian one, the profit gap normalized by the value of a direction,

$$\frac{\Pi^* - \Pi_o}{w'_o g_x + r'_o g_y}$$

which the Chambers-Chung-Fare duality splits exactly into the directional distance β in the same direction plus an allocative remainder. Both parts are non-negative and bigger is worse, matching [dea_ddf](#) and reversing the convention of the other two functions.

Constant and non-decreasing returns are refused for profit, and this is a property of the problem rather than a gap in the implementation: over a cone, one profitable reference DMU makes maximum profit unbounded, because scaling that DMU up scales its profit up with it. Maximum profit is a variable-returns concept.

An external reference set can make the cost program infeasible under variable returns, for the same reason it can for [dea_sbm](#): the program must match the evaluated DMU's output from a convex combination of the reference DMUs, and an outsider producing more than any of them cannot be matched. NA is returned with a warning naming the cause. Under constant returns the cone always reaches, so this does not arise.

Prices are assumed exogenous to the DMU. Nothing here tests that, and where a DMU is large enough to move its own prices the measures answer a question about a price-taker that the DMU is not.

Value

An object of class "dea_price", a list whose components include:

eff	Overall efficiency. For cost and revenue, a ratio in (0,1] where 1 is best. For profit, the Nerlovian gap, where 0 is best and larger is worse .
technical, allocative	The two factors of eff. Cost and revenue efficiency <i>multiply</i> : <code>eff == technical * allocative</code> . The Nerlovian gap <i>adds</i> : <code>eff == technical + allocative</code> .
optimal, observed	Minimum cost and observed cost, or maximum revenue and observed revenue. Not present for <code>dea_profit()</code> , which reports <code>profit_max</code> and <code>profit_obs</code> instead.
optimal_q	The cost-minimizing input mix, or the revenue-maximizing output mix, in the caller's units. Also returned by <code>fitted()</code> .
lambda, status, dmu, ref, n, nref, p, q	As in dea .

References

Farrell, M. J. (1957). The measurement of productive efficiency. *Journal of the Royal Statistical Society A* 120, 253–281.

Chambers, R. G., Chung, Y. and Fare, R. (1998). Profit, directional distance functions, and Nerlovian efficiency. *Journal of Optimization Theory and Applications* 98, 351–364.

See Also

[dea](#), [dea_ddf](#), [dea_cross](#)

Examples

```
X <- as.matrix(charnes1981[, paste0("x", 1:5)])
Y <- as.matrix(charnes1981[, paste0("y", 1:3)])

## One price list faced by every school site.
ce <- dea_cost(X, Y, w = c(1.4, 0.9, 2.1, 1.2, 1.0), rts = "crs")
ce

## The decomposition is exact, and this is what it buys: sites that are on the
## frontier but are not buying the cheapest mix that would keep them there.
sum(ce$technical > 1 - 1e-9)      # technically efficient
sum(ce$eff == 1)                  # AND allocatively efficient

## Profit efficiency needs prices on both sides, and variable returns.
pe <- dea_profit(X, Y, w = c(1.4, 0.9, 2.1, 1.2, 1.0), r = c(3, 2.5, 1.8))
all.equal(unname(pe$eff), unname(pe$technical + pe$allocative))
```

dea_cross

*Cross-efficiency***Description**

Peer appraisal instead of self-appraisal. Ordinary DEA lets every DMU pick the weights that flatter it most, which is why so many of them score 1 and why the ranking is so often empty. Cross-efficiency scores each DMU under *every other* DMU's optimal weights and averages the result.

Usage

```
dea_cross(x, y, data = NULL,
          secondary = c("benevolent", "aggressive", "none"),
          rts = c("crs", "vrs"),
          self = TRUE, scaling = TRUE,
          xref = NULL, yref = NULL, dataref = NULL)

## S3 method for class 'dea_cross'
plot(x, sort = TRUE, ...)
```

Arguments

x, y, data	Inputs and outputs, in any of the forms accepted by dea .
secondary	How to choose among a rater's alternate optimal weights. "benevolent" maximizes the average cross-efficiency it awards everyone else, "aggressive" minimizes it, and "none" takes whatever the solver returned. See Details : "none" is not reproducible and is not the default.
rts	Returns to scale. "vrs" is accepted only with secondary = "none".
self	Whether a DMU's own appraisal counts in its mean. FALSE drops the diagonal, which is Doyle and Green's convention.
sort	For plot, order the DMUs by their cross-efficiency. FALSE keeps the data order.
...	Passed to the underlying plotting call.
scaling, xref, yref, dataref	As in dea . The reference set is the panel of <i>raters</i> ; separating it from x/y lets a fixed panel score newcomers.

Details

$$E_{do} = \frac{u'_d y_o}{v'_d x_o}$$

is DMU *o*'s efficiency priced at DMU *d*'s own optimal weights. The column mean is *o*'s cross-efficiency. It very rarely ties, which is the practical reason to use it: on the 70 sites of [charnes1981](#) the constant-returns model calls 19 of them efficient and cannot rank them, while cross-efficiency ranks all 70.

The weights are not unique, and that is the whole difficulty. An efficient DMU generally has an entire face of optimal weight vectors, all giving it the same score of 1 but giving *other* DMUs quite different cross-efficiencies. A cross-efficiency computed from whichever vertex the solver happened to stop at is therefore not reproducible across solvers, let alone across packages, and the spread is first-order rather than numerical.

Doyle and Green (1994) resolve this by holding the rater's own score at θ_d and choosing among the remaining optima with a stated secondary goal. Reporting both the benevolent and the aggressive answer brackets the result: a ranking that survives from one to the other is a property of the data, and one that does not was a property of the solver. That is why "none" is available but is not the default.

Constant returns is the default for a reason. Under variable returns the ratio carries the intercept u_0 , is no longer bounded by 1 and can be negative, so the average of a column is not an efficiency. `dea_cross()` allows `rts = "vrs"` for description but refuses to attach a secondary goal to it, because "maximize everyone else's average" would then be optimizing something that is not an efficiency.

Value

An object of class "dea_cross", a list whose components include:

<code>eff</code>	The cross-efficiency of each DMU: the mean of the appraisals it receives. NA, with a warning, for a DMU that receives no usable appraisal — which happens when every rater's weights value its inputs at zero, most often because its inputs are all zero and the ratio has no denominator. A DMU appraised by only some of the panel is averaged over those raters, and that too is warned about rather than left to be inferred from <code>cross_matrix</code> .
<code>cross_matrix</code>	The $n_{ref} \times n$ matrix E , one row per rater and one column per rated DMU. Its diagonal is the ordinary DEA score.
<code>own</code>	Each DMU's ordinary DEA score, or NULL when the rated DMUs are not among the raters.
<code>maverick</code>	Doyle and Green's index, $(own - eff) / eff$: how much a DMU's self-appraisal exceeds what its peers award it. Large means a DMU that looks efficient only under weights nobody else would choose.
<code>spread</code>	The range of appraisals each DMU receives – a direct measure of how much the answer depends on whose weights are used. NA where <code>eff</code> is.
<code>v, u, u0</code>	The weights each rater used.

References

- Sexton, T. R., Silkman, R. H. and Hogan, A. J. (1986). Data envelopment analysis: critique and extensions. *New Directions for Program Evaluation* 32, 73–105.
- Doyle, J. and Green, R. (1994). Efficiency and cross-efficiency in DEA: derivations, meanings and uses. *Journal of the Operational Research Society* 45, 567–578.

See Also

[dea, multipliers](#)

Examples

```
X <- as.matrix(charnes1981[, paste0("x", 1:5)])
Y <- as.matrix(charnes1981[, paste0("y", 1:3)])
rownames(X) <- rownames(Y) <- charnes1981$site

ben <- dea_cross(X, Y, secondary = "benevolent")
agg <- dea_cross(X, Y, secondary = "aggressive")
summary(ben)

## The bracket. Where it is wide, the ranking is an artefact of the weights.
range(ben$eff - agg$eff)

## 19 sites tie at 1 under ordinary DEA; none tie under cross-efficiency.
sum(ben$own == 1)
length(unique(round(ben$eff, 8)))
```

dea_ddf	<i>Directional distance function</i>
---------	--------------------------------------

Description

The directional technology distance function of Chambers, Chung and Fare (1996): the number of units of a chosen direction vector by which a DMU could move toward the frontier, contracting inputs and expanding outputs at the same time.

Usage

```
dea_ddf(x, y, data = NULL,
        direction = c("both", "in", "out", "unit", "mean", "range"),
        rts = c("vrs", "crs", "nirs", "ndrs"),
        peers = TRUE, scaling = TRUE,
        xref = NULL, yref = NULL, dataref = NULL)
```

Arguments

x, y, data	Inputs and outputs, in any of the forms accepted by dea . Unlike the other models here, values may be zero or negative.
direction	The direction $g = (g_x, g_y)$ along which distance is measured. One of "both" $g = (x_o, y_o)$, each DMU's own data. The default. "in" $g = (x_o, 0)$; then $\beta = 1 - \theta$. "out" $g = (0, y_o)$; then $\beta = \phi - 1$. "unit" $g = (1, \dots, 1)$, one unit of every variable. "mean" the sample mean of each variable, the same for all DMUs. "range" $g_o = (x_o - \min x, \max y - y_o)$, the vector from the DMU to the technology's ideal point. This is the range directional model of Portela, Thanassoulis and Simpson (2004); see 'The range directional model' below. The minimum and maximum are taken over the <i>reference</i> set, unlike "mean", which uses the evaluated set.

Or a numeric vector of length $p + q$ giving one fixed direction, inputs first; or a n by $p + q$ matrix giving one per DMU.

rts Returns to scale, as in [dea](#).

peers, scaling, xref, yref, dataref As in [dea](#).

Details

Why not just use a radial measure. Three reasons, in order of how often they bite.

Inputs and outputs move together. A radial measure has to pick a side; "both" contracts inputs and expands outputs at once, which is what most efficiency questions actually ask.

Zeros and negatives are fine. β is an additive contraction, so nothing is divided by a DMU's own level. A zero output leaves the radial output score undefined and the slacks-based measure undefined; here it is just a number. Net income, net exports and abatement credits can be negative, and the directional distance function is the standard way to handle them.

The direction is visible. A radial model also chooses a direction; it simply does not say which.

Units. β is scale invariant when g is proportional to the DMU's own data and is *not* when g is fixed, since doubling an input's units halves the β needed to move one unit of g . That is a property of the estimand rather than a defect, but a fixed direction must be stated in the same units as the data. Internal rescaling is applied to the direction as well, so the reported β is the one the caller's units imply.

No common ratio scale. `efficiency(fit, "score")` refuses to convert β to a $(0, 1]$ score for a direction that is not purely input or purely output, because no such conversion exists.

The range directional model. `direction = "range"` is the model of Portela, Thanassoulis and Simpson (2004). The direction is the vector from each DMU to the *ideal point* of the reference set — the corner of its bounding box that is best in every coordinate at once — so β reads as the fraction of the distance to that point the DMU could travel. Two properties follow, and both are `rts = "vrs"` properties.

β lies in $[0, 1]$. Under variable returns $X'\lambda$ is a convex combination of the reference inputs and so cannot fall below their minimum, which forces $\beta \leq 1$; the same argument applies on the output side. The bound also held under every cone tested here, but the elementary argument does not cover them, so it is stated for what it is proved for.

β is translation invariant, which is the reason to use this model at all. Adding a constant to a column leaves g unchanged, since it is a difference of two values in that column, and shifts both sides of the constraint by $t \sum \lambda$ — equal only when $\sum \lambda = 1$. So negative inputs or outputs need no translation before fitting, and the answer does not depend on one. Under a cone this fails, and by a wide margin rather than subtly.

The range is taken over the *reference* set, because it is the reference DMUs that span the technology and the $[0, 1]$ bound is a statement about that technology. With a separate `xref/yref` an evaluated DMU can lie outside that set's box; a direction component that would point backwards is clamped to zero, with a warning, since a coordinate already better than anything in the technology has no room to improve. A DMU that attains the best value in *every* input and output is the ideal point itself: its direction is zero, its β is 0, no program is solved — it would be unbounded, not infeasible — and its peers are reported as NA, with a warning.

`dea_profit` takes the same argument, but has no answer at such a DMU: the Nerlovian measure divides the profit gap by $w'g_x + r'g_y$, which is zero there.

Value

An object of class "dea" with model = "ddf". Note that eff (also available as beta) measures *inefficiency*: **0 is on the frontier and larger is worse**, the reverse of every other model in this package. The direction actually used is returned in g. There are no second-stage slacks – the projection is along g .

A DMU whose program does not solve is reported as NA with a warning, and its status says which kind of failure it was: 2 for an infeasible program, anything other than 0 or 1 for a numerical one. Infeasibility is reachable in ordinary use — direction = "in" holds g_y at zero, so a DMU whose outputs exceed everything the reference set can produce has no feasible point at all.

References

- Chambers, R. G., Chung, Y. and Fare, R. (1996). Benefit and distance functions. *Journal of Economic Theory* 70, 407–419.
- Chung, Y. H., Fare, R. and Grosskopf, S. (1997). Productivity and undesirable outputs: a directional distance function approach. *Journal of Environmental Management* 51, 229–240.
- Portela, M. C. A. S., Thanassoulis, E. and Simpson, G. (2004). Negative data in DEA: a directional distance approach applied to bank branches. *Journal of the Operational Research Society* 55, 1111–1121.

See Also

[dea](#), [dea_sbm](#)

Examples

```
sim <- dea_sim(60, p = 2, q = 1, seed = 1)

dea_ddf(sim$x, sim$y, direction = "both", rts = "vrs")

## The radial models are the two one-sided directions.
b <- dea_ddf(sim$x, sim$y, direction = "in", rts = "vrs")
t <- dea(sim$x, sim$y, orientation = "in", rts = "vrs", slack = FALSE)
max(abs(b$beta - (1 - t$eff)))

## A fixed direction: one unit of every variable.
dea_ddf(sim$x, sim$y, direction = "unit", rts = "vrs")

## The range directional model: beta is the fraction of the distance to the
## ideal point, so it lies in [0, 1], and it survives data of either sign.
r <- dea_ddf(sim$x, sim$y, direction = "range", rts = "vrs")
range(r$beta)
shifted <- dea_ddf(sim$x - 3, sim$y - 1, direction = "range", rts = "vrs")
max(abs(shifted$beta - r$beta)) # translation invariant under vrs
```

dea_malmquist

*Panel data and the Malmquist productivity index***Description**

`dea_panel()` holds a long-format panel — one row per DMU per period — and is the layer every index over time is built on. `dea_malmquist()` computes the Malmquist productivity index of Fare, Grosskopf, Norris and Zhang (1994) on it, for each pair of consecutive periods, with its decomposition into efficiency change and technical change and the further split of efficiency change into a pure and a scale part.

Usage

```
dea_panel(x, y, id, period, data = NULL)

dea_malmquist(x, y = NULL, id = NULL, period = NULL, data = NULL,
              rts = c("crs", "vrs", "nirs", "ndrs"),
              orientation = c("in", "out"),
              scale_decomposition = TRUE,
              scaling = TRUE)
```

Arguments

<code>x, y</code>	Inputs and outputs, stacked over DMU-periods, in any of the forms dea accepts. For <code>dea_malmquist()</code> , <code>x</code> may instead be a "dea_panel" object, in which case <code>y</code> , <code>id</code> and <code>period</code> are taken from it.
<code>id</code>	Which DMU each row belongs to: a vector, or — with <code>data</code> — a column name or a one-sided formula. Labels, not numbers: they are compared, never arithmetic.
<code>period</code>	Which period each row belongs to, in the same forms. The order matters ; see Details .
<code>data</code>	Optional data frame the column names or formulas refer to.
<code>rts</code>	Returns to scale. The default "crs" is the technology the index is defined on; see Details before changing it.
<code>orientation</code>	"in" or "out". Both report $M > 1$ as growth.
<code>scale_decomposition</code>	Split efficiency change into a pure and a scale part. Defined only for the "crs" index and ignored, with a warning if it was asked for explicitly, otherwise.
<code>scaling</code>	As in dea .

Details

The four distance functions, and what their names mean. For a DMU observed in periods t ("from") and $t + 1$ ("to"), the index needs that DMU scored against both periods' technologies:

d_ff	the from-period observation on the from-period technology
d_tt	to on to
d_tf	the to-period observation on the <i>from</i> -period technology
d_ft	from on <i>to</i>

The convention is **observation first, technology second**, and it is worth stating because the two mixed terms are easy to swap and swapping them replaces efficiency change with its reciprocal. Each is an ordinary *dea* fit with *xref*/*yref* pointing at the other period; the separation of the evaluated and reference sets is what makes this possible at all. Then

$$EC = d_{tt}/d_{ff}, \quad TC = \sqrt{(d_{tf}/d_{tt})(d_{ff}/d_{ft})}, \quad M = EC \times TC.$$

Every score is put on the common $(0, 1]$ efficiency scale first, so the output-oriented case converts ϕ to $1/\phi$ once at the source rather than carrying a second set of formulas. The mixed terms routinely leave $(0, 1]$ — an observation scored against another period's frontier need not lie inside it — and that is the measurement rather than an error.

A convention the field is split on. This package follows Fare et al.'s: the index is built from Shephard distance functions, so $M > 1$ is productivity growth whichever side it is measured from, and the two orientations agree under "crs". `productivity::malm()` reports the same numbers in both orientations. `Benchmarking::malmq()` instead builds the index from the native scale of each orientation, so **its output-oriented index is the reciprocal of this one**. Neither is wrong; they are different conventions, and a number copied between them without checking is.

Why the default is constant returns. The mixed distance functions can be *infeasible* under variable returns: a convex hull cannot extrapolate to a point outside it, while a cone can always be scaled to reach one. That is why the index is conventionally computed on a cone. "vrs" is allowed here; the infeasible terms come back NA and the count is reported per period pair rather than left to be discovered.

The pure/scale split is an identity, so it is worth saying what it does and does not establish. With v the variable-returns own-period scores,

$$PEC = v_{tt}/v_{ff}, \quad SEC = (d_{tt}/v_{tt})/(d_{ff}/v_{ff}),$$

and $EC = PEC \times SEC$ follows by cancellation — so that product cannot fail and checking it establishes nothing. What the package's tests check is that each piece equals the same quantity computed from independent variable- and constant-returns fits.

The order of the periods is data, not a guess. A factor's levels are honoured as given; anything else is sorted, which is right for numbers and for ISO-like dates and *wrong* for labels such as "Q1", "Q10", "Q2". Reversing the order inverts every index, so the panel object prints the order it settled on and a factor is how to override it.

An unbalanced panel is an ordinary case. Each period pair uses the DMUs observed in both; anything seen in only one has no index for that pair. `summary()` on the panel reports how many DMUs each pair has and how many observations went unmatched.

The technology is estimated per period, from that period's DMUs alone. Pooling the periods would define the frontier shift away.

Value

`dea_panel()` returns an object of class "dea_panel" with the stacked X and Y , the `id` and period of each row, the periods in the order they will be used, and whether the panel is balanced.

`dea_malmquist()` returns an object of class "dea_malmquist". Its table has one row per DMU per consecutive period pair, with columns `id`, `from`, `to`, `malmquist`, `effch`, `techch` and — under `rts = "crs"` — `purech` and `scalech`. `distances` carries the four distance functions each row was built from. A value above 1 is growth in every column.

References

Fare, R., Grosskopf, S., Norris, M. and Zhang, Z. (1994). Productivity growth, technical progress, and efficiency change in industrialized countries. *American Economic Review* 84, 66–83.

Caves, D. W., Christensen, L. R. and Diewert, W. E. (1982). The economic theory of index numbers and the measurement of input, output, and productivity. *Econometrica* 50, 1393–1414.

See Also

[dea](#), [dea_rts](#)

Examples

```
## A two-period panel with a frontier that moves outward.
a <- dea_sim(25, p = 2, q = 1, seed = 1)
b <- dea_sim(25, p = 2, q = 1, seed = 2)
pan <- dea_panel(rbind(a$x, b$x * 0.92), rbind(a$y, b$y * 1.12),
                id = rep(1:25, 2), period = rep(c(2023, 2024), each = 25))
pan

m <- dea_malmquist(pan, rts = "crs")
m
head(m$table)

## The four distances the index was built from.
head(m$distances)

## Column names work too, when the panel is a data frame.
df <- data.frame(firm = rep(1:25, 2), year = rep(c(2023, 2024), each = 25),
                k = pan$X[, 1], l = pan$X[, 2], q = pan$Y[, 1])
dea_malmquist(c("k", "l"), "q", "firm", "year", data = df, rts = "crs")
```

dea_mult

The multiplicative, or piecewise Cobb-Douglas, model

Description

Envelop the data with piecewise *log*-linear facets rather than linear ones, so that each facet is a Cobb-Douglas function and a peer group is a weighted geometric mean. The multiplicative model of Charnes, Cooper, Seiford and Stutz (1982, 1983).

Usage

```
dea_mult(x, y, data = NULL, rts = c("vrs", "crs", "nirs", "ndrs"),
        peers = TRUE, xref = NULL, yref = NULL, dataref = NULL)

## S3 method for class 'dea_mult'
print(x, ...)
## S3 method for class 'dea_mult'
summary(object, ...)
```

Arguments

<code>x, y, data</code>	Inputs and outputs, in any of the forms accepted by dea . Every value must be strictly positive : the model takes logs. For the print method, an object of class "dea_mult".
<code>rts</code>	Restriction on $\sum \lambda$, <i>in log space</i> . Only "vrs" gives a units-invariant model; the others warn. See below.
<code>peers, xref, yref, dataref</code>	As in dea .
<code>object</code>	An object of class "dea_mult".
<code>...</code>	Ignored.

Details

Every other estimator here builds its technology from *convex* combinations, so the frontier is piecewise linear and a peer target is an arithmetic average of its peers. That is an assumption about the production function, and it is usually made silently. This model makes the other one:

$$\text{target } x_i = \prod_j x_{ij}^{\lambda_j} \quad \text{rather than} \quad \sum_j \lambda_j x_{ij}$$

so each facet of the frontier is a Cobb-Douglas function $y = A \prod_i x_i^{b_i}$ whose exponents are estimated rather than assumed. For most production studies that is the natural shape. No other package on CRAN implements it.

It needs no new solver: taking logs turns a product into a sum, so the program is exactly the unweighted additive model of [dea_add](#) on $\log(x)$ and $\log(y)$. The slacks are then log ratios, and `ratio_x` and `ratio_y` are their exponentials, reported so that nobody has to remember to exponentiate.

Units invariance needs `rts = "vrs"`. Multiplying an input column by c adds $\log c$ to that column in log space, so units invariance here *is* the additive model's translation invariance (Ali and Seiford 1990), which needs the convexity row. The mean objective on a 40-DMU design with one input column multiplied by c :

rts	c = 1	c = 10	c = 1000
vrs	0.370673	0.370673	0.370673
crs	0.441869	0.950590	2.187686
nirs	0.424713	0.949094	2.187686
ndrs	0.387829	0.372169	0.370673

Under "vrs" nothing moves; under "crs" the answer changes by a factor of five although nothing was done but record an input in different units. Geometrically, without the convexity row the log-space technology is a cone through the log-space origin — the point $x = y = 1$, which has no meaning and moves whenever a unit changes.

The other technologies are still offered, with a warning, because "crs" is the published 1982 model and the 1983 paper's title ("Invariant multiplicative efficiency...") says what it was written to fix; reproducing a published result is a real use. That is the opposite of the choice made in [dea_undesirable](#), where the invalid combinations are refused outright — there, no one had proposed them, so serving them would have invented a quantity.

rts does not mean returns to scale here. These are restrictions on $\sum \lambda$ in *log* space. "vrs" does give a variable-returns Cobb-Douglas envelope, but "crs" does not give a constant-returns technology in the original space; it gives the cone described above. The argument keeps its name because it is the same row of the same program, not because the returns-to-scale reading survives the transformation.

Value

An object of class `c("dea_mult", "dea")`, so the methods for "dea" apply. Its own components:

`eff` $\exp(-\text{total log slack})$, in $(0, 1]$ with 1 exactly on the frontier. It equals $\prod_i (x_i^*/x_{io}) \prod_r (y_{ro}/y_r^*)$.

`ratio_x, ratio_y` The factor each input could be multiplied by (in $(0, 1]$) and each output could be multiplied by (in $[1, \infty)$). These are the model's native quantities.

`log_slack_x, log_slack_y` The underlying additive slacks in log space.

`slack_x, slack_y` The same improvements as differences in the caller's units, so that code written against "dea" still reads.

The bias runs the other way when the technology is not log-convex

This is the one thing to understand before using the model in place of a linear-envelope one.

The whole of DEA's inferential story rests on the estimated frontier lying *inside* the true one, so that every score is biased toward 1 in a known direction — that is what [dea_boot](#) corrects. For this model that holds only if the true technology is **log-convex**, i.e. if $\log f(e^z)$ is concave. It is not implied by ordinary convexity.

For a linear frontier $f(x) = 0.5x_1 + 0.5x_2$, $\log f(e^z)$ is log-sum-exp, which is *convex*, so the log chord runs above the surface: between $x = (2, 9)$ and $(9, 2)$ the chord gives 5.50 where the truth is 4.24 at the midpoint, 30% high. No estimator is involved in that, it is a property of the hull.

Measured in `horserace/multiplicative_experiment.R`, which asks both envelopes for $\max\{y : (x_0, y) \in T\}$ at the same fixed interior grid, so that only the envelope differs: 100 replications at $n = 150$:

true shape	envelope	mean signed shortfall	% of points above the truth
cobb	log-convex	0.0089	0.0
cobb	convex	0.0211	0.0
linear	log-convex	-0.0804	97.1

linear	convex	0.0091	0.0
ces	log-convex	0.0174	0.0
ces	convex	0.0239	0.0

On a technology that really is Cobb-Douglas the log-convex envelope is 2.4 times tighter and still inside; on a CES technology, which is neither shape, it is 1.4 times tighter and still inside. On a *linear* technology it sits above the truth at 97% of the evaluation points. That is not a larger error but a different kind of error: the estimated technology contains points the true one does not.

And it does not improve with more data — it cannot, because the estimate is converging to the log-convex hull of the technology, which strictly contains it. Between $n = 80$ and $n = 150$ the overshoot grew from -0.046 to -0.080 and the share of violated points from 87.5% to 97.1%. More data makes this model more confidently wrong about the wrong technology.

So this is a maintained hypothesis about the production function, not a drop-in alternative. Use it when a Cobb-Douglas or log-convex technology is what you would have assumed anyway, and do not read a score from it as conservative.

References

- Ali, A. I. and Seiford, L. M. (1990). Translation invariance in data envelopment analysis. *Operations Research Letters* 9, 403–405.
- Charnes, A., Cooper, W. W., Seiford, L. and Stutz, J. (1982). A multiplicative model for efficiency analysis. *Socio-Economic Planning Sciences* 16, 223–224.
- Charnes, A., Cooper, W. W., Seiford, L. and Stutz, J. (1983). Invariant multiplicative efficiency and piecewise Cobb-Douglas envelopments. *Operations Research Letters* 2, 101–103.

See Also

[dea_add](#), which is the same program on untransformed data, [dea](#), [dea_boot](#).

Examples

```
set.seed(3)
n <- 40
x <- matrix(runif(n * 2, 2, 10), n, 2,
             dimnames = list(NULL, c("labour", "capital")))
y <- matrix((x[, 1] * x[, 2])^0.4 * exp(-abs(rnorm(n, 0, 0.25)))), n, 1,
             dimnames = list(NULL, "output"))

m <- dea_mult(x, y)
m
head(m$ratio_x)      # the factor each input could be multiplied by

## The target is a weighted GEOMETRIC mean of the peers.
all.equal(unname(x * m$ratio_x), unname(exp(m$lambda %*% log(x))),
          tolerance = 1e-8)

## And the score does not change when an input is recorded in other units.
x2 <- x; x2[, 1] <- x2[, 1] * 1000
all.equal(dea_mult(x2, y)$eff, m$eff, tolerance = 1e-9)
```

dea_nd

*Non-discretionary inputs and outputs***Description**

Radial efficiency when some inputs or outputs are outside the DMU's control: they still constrain who may be a peer, but no credit is given for changing them. Banker and Morey (1986a).

Usage

```
dea_nd(x, y, data = NULL, nd_x = NULL, nd_y = NULL,
      rts = c("vrs", "crs", "nirs", "ndrs"),
      orientation = c("in", "out"),
      slack = TRUE, peers = TRUE, scaling = TRUE,
      xref = NULL, yref = NULL, dataref = NULL)
```

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

Arguments

x, y, data	Inputs and outputs, in any of the forms accepted by dea . For the print method, an object of class "dea_nd".
nd_x, nd_y	Which inputs and outputs are non-discretionary, as a logical mask, column positions, or column names. NULL means none, and the fit is then exactly dea 's.
rts, orientation, slack, peers, scaling, xref, yref, dataref	As in dea .
...	Ignored.

Details

Some of a DMU's inputs are not its to choose. A school cannot vary the number of pupils it is sent; a hospital cannot choose its catchment. The ordinary radial program contracts every input by the same θ , so it reports a unit as inefficient for failing to do something it has no power to do.

The change is one row type:

$$\sum_j \lambda_j x_{ij} \leq \theta x_{io} \quad i \text{ discretionary}$$

$$\sum_j \lambda_j x_{ij} \leq x_{io} \quad i \text{ fixed}$$

so a fixed input still constrains who may be a peer — a unit with twice the pupils is not a fair comparison — while no credit is given for cutting it.

The two stages mean different things. Stage one scales only the oriented side, so only nd_x changes an input-oriented score and only nd_y an output-oriented one. Stage two maximises a sum

of slacks, and there *both* masks apply in *both* orientations: a slack the DMU cannot act on should earn it no credit whichever way the program is oriented. That is Banker and Morey's own rule — the non-discretionary slacks are dropped from the objective and kept in the constraints — and it is why `nd_y` is not ignored under an input orientation even though it leaves θ alone.

Value

An object of class `c("dea_nd", "dea")`, with `nd_x` and `nd_y` as named logical masks and `n_fixed` counting them. `efficient` is Pareto-Koopmans over the *discretionary* coordinates only: residual slack on a fixed variable is not something the DMU failed to do.

Fixing a variable makes the measured inefficiency larger

Every intuition says otherwise, so it is worth stating flatly: $\theta_{nd} \leq \theta_{ordinary}$, always. Measured over 8704 DMU-fits across four technologies and random dimensions, the largest excess of θ_{nd} over $\theta_{ordinary}$ was 1.2e-12, which is solver noise; on the example below the mean θ falls from 0.901 to 0.789. Output oriented the mirror holds, $\phi_{nd} \geq \phi_{ordinary}$.

The reason is one line about the feasible set. For $\theta < 1$ the fixed row $\sum_j \lambda_j x_{ij} \leq x_{io}$ is *looser* than $\leq \theta x_{io}$, so the non-discretionary program's feasible set contains the ordinary one and its minimum can only fall. Economically: θ in the ordinary model is a single proportional contraction of every input at once, so a peer must use proportionally less of everything, pupils included — a demanding requirement. Here a peer need only use no more pupils, and the whole contraction lands on the teachers.

So the two are not the same quantity measured more or less fairly. They answer different questions — "could all inputs have been smaller together?" against "with these pupils, could the teachers have been fewer?" — and a table comparing them as though one were a corrected version of the other is comparing two estimands.

Non-discretionary outputs are not the mirror image

The symmetry looks exact and is not, and chapter 21 of Charnes, Cooper, Lewin and Seiford (1994) flags the confusion as a recurring interpretational error.

"Non-discretionary input" has a clean meaning: a quantity the DMU is handed. "Non-discretionary output" is usually used for something quite different — an output the DMU influences only *indirectly*, through its inputs, such as sales through advertising. That is not fixed; it is controlled at one remove, and this formulation does not model it. Marking it `nd_y` tells the program the DMU should get no credit for producing more of it, which for sales is plainly wrong.

Use `nd_y` only for an output that is genuinely exogenous — a statutory quota, a count of mandated cases. If the right description is "controlled indirectly", neither this nor any other model in this package says so, and `dea_nd()` prints a message whenever `nd_y` is used so the distinction is made deliberately.

References

- Banker, R. D. and Morey, R. C. (1986a). Efficiency analysis for exogenously fixed inputs and outputs. *Operations Research* 34, 513–521.
- Charnes, A., Cooper, W. W., Lewin, A. Y. and Seiford, L. M. (1994). *Data Envelopment Analysis: Theory, Methodology and Application*. Kluwer.

See Also

[dea_categorical](#) for an environment that is ordered rather than measured, [dea](#).

Examples

```
set.seed(31)
n <- 60
teachers <- runif(n, 10, 40)
pupils <- runif(n, 150, 600) # sent to the school, not chosen by it
x <- cbind(teachers = teachers, pupils = pupils)
y <- matrix(teachers^0.3 * pupils^0.6 * exp(-abs(rnorm(n, 0, 0.2))), n, 1,
            dimnames = list(NULL, "attainment"))

r <- dea_nd(x, y, nd_x = "pupils")
r

## Lower than the ordinary fit, which is the direction to expect.
c(ordinary = mean(dea(x, y)$eff), non_discretionary = mean(r$eff))
```

dea_reg

*Two-stage efficiency regression (Simar and Wilson 2007)***Description**

Regress efficiency on covariates, correctly. `dea_reg()` implements Algorithms #1 and #2 of Simar and Wilson (2007): a truncated regression of the estimated distance functions on z , with a bootstrap for the standard errors, and — under Algorithm #2 — a bias correction of the scores before the regression is run at all.

Usage

```
dea_reg(object, z, data = NULL,
        algorithm = c("sw2", "sw1"),
        L1 = 100L, L2 = 2000L, alpha = 0.05,
        seed = NULL, progress = interactive())
```

Arguments

<code>object</code>	A radial fit from dea .
<code>z</code>	The covariates, in any of the forms dea accepts for x : a matrix, a data frame, column names or a one-sided formula. An intercept is added unless a constant column is already present.
<code>data</code>	Optional data frame the names or formula refer to.
<code>algorithm</code>	"sw2" for the double bootstrap of Algorithm #2, the default and the one the paper recommends; "sw1" for the single bootstrap of Algorithm #1.
<code>L1</code>	Replications in the bias-correction loop, Algorithm #2 only. Each one is a full DEA sweep, so this is where the time goes.

L2	Replications in the bootstrap of the coefficients.
alpha	One minus the nominal coverage of the intervals.
seed	Passed to <code>set.seed()</code> for reproducibility.
progress	Print a counter.

Details

Why not OLS, and why not Tobit. Regressing DEA scores on covariates is one of the most common things done with DEA and one of the most common things done wrong. The scores share an estimated frontier, so they are correlated by construction in a way that depends on the sample, and conventional standard errors do not describe their sampling distribution. Tobit does not fix this. Censoring is a statement about how a variable is *observed*; the mass at the frontier here is an artefact of a boundary estimator, not a censored observation of something else. The efficient DMUs are therefore *dropped*, not censored — once it is known the frontier was hit, that observation says nothing further about z .

The response is Shephard's distance, so read the signs carefully. The regression is in $\delta \geq 1$, with $\delta = 1/\theta$ input-oriented and $\delta = \phi$ output-oriented, truncated from the left at 1. A **positive** coefficient therefore means the covariate is associated with *less* efficiency. That is the opposite of what a reader expects from a regression on θ , it is what the paper reports, and `print()` says so on every fit.

What Algorithm #2 adds, and why it is the default. $\hat{\delta}$ is a biased estimate of δ , biased toward 1, so Algorithm #1 regresses on something systematically shifted. The bias is roughly common across DMUs, so it lands mostly on the *intercept*. Algorithm #2's inner loop draws a δ^* from the fitted model, builds a pseudo-sample whose true distance is that δ^* — scaling a DMU's inputs by a factor scales its input distance by the same factor, so the construction is exact — re-estimates the original points against the pseudo-technology, and corrects $\hat{\delta}$ before the regression is fitted.

As in [dea_boot](#), the pseudo-sample is the *technology* and the points *evaluated* against it are the originals. Getting that backwards produces a number for every DMU and measures nothing.

On a design whose β is known, 200 replications at $n = 150$, against a nominal 95% interval:

	Algorithm #1	Algorithm #2	OLS on $\hat{\delta}$
intercept, bias	-0.070	-0.039	+0.052
intercept, coverage	0.650	0.785	0.440
slope 1, bias	-0.015	+0.005	-0.160
slope 1, coverage	0.900	0.905	0.065
slope 2, bias	+0.009	-0.001	+0.083
slope 2, coverage	0.885	0.920	0.000

Read the slope rows first. **The naive route's intervals essentially never contain the truth** — 0.065 and 0.000 against a nominal 0.95 — and its slope bias is ten times Algorithm #2's. Two things are wrong with it at once and the table cannot separate them: the likelihood is misspecified, because the data generating process is truncated, and the standard errors describe a sampling distribution the scores do not have.

One cell goes the other way, and it is worth saying so. On the *intercept point estimate* OLS is no worse: its bias is +0.052 against Algorithm #2's -0.039, in the opposite direction and of much the same size, and paired across the 200 datasets the difference in absolute error does not resolve.

The case against the naive route rests on the slopes and on the intervals, not on every number in the table.

Between the two algorithms: **#2 is nearer the truth for every coefficient**, and comparing them on the same 200 datasets rather than as two samples, decisively so (paired $|t|$ of 13.4, 3.8 and 4.4). On interval *coverage* it beats #1 on the intercept, +0.135 with a paired standard error of 0.027; for the slopes both sit near 0.90 and the difference is not resolvable at this many replications. **Neither reaches nominal coverage on the intercept**, which is where the frontier bias concentrates.

The script that measures this is `horserace/twostage_experiment.R` in the package's development repository; a smaller version of it is a test.

What is not bootstrapped. The covariates. z is fixed, exactly as in the paper; the resampling is of the error term and, under Algorithm #2, of the frontier.

Cost. Algorithm #1 is L2 truncated regressions and is fast. Algorithm #2 adds L1 full DEA sweeps, which is where essentially all of its time goes; the paper's suggested $L1 = 100$ is the default.

Value

An object of class "dea_reg". `table` has one row per coefficient with the estimate, the bootstrap standard error, the interval and the bootstrap's own estimate of the bias in that coefficient. `delta` is the response the regression used, `delta_bc` and `bias` the corrected scores and the correction under Algorithm #2, and `boot` the full matrix of bootstrap replications.

References

- Simar, L. and Wilson, P. W. (2007). Estimation and inference in two-stage, semi-parametric models of production processes. *Journal of Econometrics* 136, 31–64.
- Simar, L. and Wilson, P. W. (2011). Two-stage DEA: caveat emptor. *Journal of Productivity Analysis* 36, 205–218.

See Also

[dea](#), [dea_boot](#)

Examples

```
## A technology whose true distances depend on z.
set.seed(1)
n <- 80
X <- matrix(runif(2 * n, 1, 10), n, 2)
z <- cbind(z1 = runif(n), z2 = rnorm(n))
delta <- 1 + rexp(n, 1 / (0.1 + 0.25 * z[, 1]))
y <- matrix(apply(X, 1, function(r) prod(r)^0.45) / delta, ncol = 1)

fit <- dea(X, y, rts = "vrs", orientation = "out", slack = FALSE)

## Algorithm #1 is quick; Algorithm #2 costs L1 DEA sweeps on top.
dea_reg(fit, z, algorithm = "sw1", L2 = 200, seed = 1, progress = FALSE)

r <- dea_reg(fit, z, algorithm = "sw2", L1 = 50, L2 = 500, seed = 1,
```

```
summary(r)                                progress = FALSE)
```

dea_rts	<i>Scale efficiency and returns to scale</i>
---------	--

Description

Fit the constant-, variable- and non-increasing-returns technologies to the same data in one sweep, and use the three scores to decompose efficiency into its technical and scale parts and to classify each DMU’s local returns to scale.

Usage

```
dea_rts(x, y, data = NULL, orientation = c("in", "out"), tol = 1e-6,
        scaling = TRUE)

## S3 method for class 'dea_rts'
print(x, ...)
## S3 method for class 'dea_rts'
summary(object, ...)
## S3 method for class 'dea_rts'
plot(x, ...)
```

Arguments

x, y, data	Inputs and outputs, in any of the forms accepted by dea . For the print method, an object of class "dea_rts".
orientation	"in" or "out", as in dea .
tol	How close two efficiency scores must be to count as equal. The classification is a comparison of three linear-programming optima, so it needs a tolerance; 1e-6 is loose relative to the solver and tight relative to any economically meaningful difference.
scaling	As in dea .
object	An object of class "dea_rts".
...	Ignored.

Details

Scale efficiency is $SE = \theta_{CRS} / \theta_{VRS}$ under an input orientation and ϕ_{VRS} / ϕ_{CRS} under an output orientation, so it lies in $(0, 1]$ either way. It is 1 when the DMU operates at the most productive scale size, and the shortfall is the part of its distance to the constant-returns frontier that comes from being the wrong *size* rather than from being badly run.

The classification uses the non-increasing technology, which permits scaling a reference point down but not up. A DMU in the increasing-returns region is smaller than the most productive scale

size, so the constant-returns frontier above it is reached by scaling its peer *down* – which the non-increasing technology permits, and there it coincides with the constant-returns frontier. A DMU in the decreasing-returns region would need its peer scaled *up*, which it forbids, and there it coincides with the variable-returns frontier:

$$\theta_{CRS} = \theta_{VRS} \Rightarrow \text{constant returns}$$

$$\theta_{NIRS} = \theta_{CRS} \neq \theta_{VRS} \Rightarrow \text{increasing returns}$$

$$\theta_{NIRS} = \theta_{VRS} \neq \theta_{CRS} \Rightarrow \text{decreasing returns}$$

Note that the rule runs opposite to what the label suggests: it is agreement with the *constant*-returns frontier that marks the increasing-returns region.

The often-quoted alternative – read $\sum \lambda$ off the constant-returns program and call it increasing below 1, decreasing above – gives the same answer when that program has a unique optimum and an arbitrary one when it does not, which is common. `sum_lambda_crs` is reported so the two can be compared, but the verdict uses the rule above.

The orientation changes the answer, and is meant to. Returns to scale are a property of the frontier *point* a DMU is benchmarked against, not of the DMU's own coordinates. An output-oriented projection holds inputs fixed, so the verdict describes the DMU's own scale. An input-oriented projection moves inputs down, and for a badly inefficient DMU it can land below the most productive scale size even though the DMU itself sits above it — so the same DMU can be classified decreasing by the output sweep and increasing by the input one. Both are correct about their own projection. Choose the orientation that matches the decision being made, and do not read the two tables against each other row by row.

These are point estimates from an estimator with a known bias, and the classification inherits that: a DMU whose constant- and variable-returns scores differ by 0.001 is classified with exactly the confidence of one whose scores differ by 0.4. For a test of the sample as a whole, with a p-value attached, see [dea_rts_test](#) — and read its warning about the input orientation, which applies to the `scale_eff` column here too.

Value

An object of class "dea_rts" whose table component is a data frame with one row per DMU and columns `dmu`, `crs`, `vrs`, `nirs`, `scale_eff`, `rts` (one of "irs", "crs", "drs") and `sum_lambda_crs`.

References

- Coelli, T. J., Rao, D. S. P., O'Donnell, C. J. and Battese, G. E. (1998). *An Introduction to Efficiency and Productivity Analysis*. Springer.
- Fare, R., Grosskopf, S. and Lovell, C. A. K. (1994). *Production Frontiers*. Cambridge University Press.
- Simar, L. and Wilson, P. W. (2002). Non-parametric tests of returns to scale. *European Journal of Operational Research* 139, 115–132.

See Also

[dea_rts_test](#) for the bootstrap test, [dea](#), [dea_boot](#)

Examples

```
sim <- dea_sim(80, p = 1, q = 1, returns = 0.7, seed = 2)
r <- dea_rts(sim$x, sim$y)
r
head(r$table)
```

dea_rts_test

Nonparametric test of returns to scale

Description

Test whether the technology as a whole may be taken to be constant-returns (or non-increasing-returns) against the variable-returns alternative, by the bootstrap of Simar and Wilson (2002). Where [dea_rts](#) classifies each DMU with no standard error attached, this attaches a p-value to a statement about the sample.

Usage

```
dea_rts_test(x, y, data = NULL, h0 = c("crs", "nirs", "ndrs"),
             orientation = c("in", "out"), B = 2000, alpha = 0.05,
             bw = c("silverman", "nrd0", "ucv", "sj"), seed = NULL,
             ncores = 1L, progress = interactive(), scaling = TRUE)

## S3 method for class 'dea_rts_test'
print(x, ...)
## S3 method for class 'dea_rts_test'
summary(object, ...)
```

Arguments

x, y, data	Inputs and outputs, in any of the forms accepted by dea . For the print method, an object of class "dea_rts_test".
h0	The technology under the null: "crs", "nirs" or "ndrs". The alternative is always "vrs". Benchmarking's spellings "drs" and "irs" are accepted, as elsewhere in this package.
orientation	"in" or "out", as in dea . Read What this test cannot be asked below before choosing "in".
B	Bootstrap replications. The p-value's resolution is $1/(B + 1)$, so a small B is not merely imprecise, it is a floor on how small a p-value can be reported; B below 200 warns.
alpha	Nominal level, used for reject and for the simultaneous level applied to the per-DMU tests.
bw	Bandwidth for the kernel that smooths the resampled efficiencies, as in dea_boot , or a single positive number.

seed	Integer seed. Set and restored, so the caller's random stream is left as it was found.
ncores	Replications to run in parallel, as in dea_boot .
progress	Print a replication counter.
scaling	As in dea . The statistic is units invariant either way; this only conditions the programs.
object	An object of class "dea_rts_test".
...	Ignored.

Details

Under the null the restricted technology is the true one, so the restricted and the unrestricted estimator are consistent for the same distance and their ratio tends to 1. Under the alternative the restricted technology is strictly larger than the truth, the restricted scores are strictly worse, and the ratio tends to something below 1. So **the test is one-sided and rejects for small values**. Two aggregations of the per-DMU ratio are reported, both from Simar and Wilson (2002, section 4): the ratio of the two means, and the mean of the per-DMU ratios. They are different numbers and need not move together — the first weights a DMU by its own level and the second does not.

The scale the statistic is written on. Scores enter on this package's common $(0, 1]$ scale — θ input-oriented, $1/\phi$ output-oriented, which is what `efficiency(fit, "score")` returns. That makes the statistic lie in $(0, 1]$ and the test left-tailed in both orientations, and makes `mean_of_ratios` exactly the mean of [dea_rts](#)'s `scale_eff` column.

It also settles something the two packages on CRAN with this test settle differently. Output-oriented, `npsf::nptestrts` aggregates on the ϕ scale while this package and `rDEA::rts.test` aggregate on the $1/\phi$ scale. On the 40-DMU design used for the recorded reference values:

	this package / rDEA	npsf	1 / npsf
ratio of means	0.858356	1.221348	0.818767
mean of ratios	0.862094	1.244737	0.803382

The third column is not the first. A mean does not commute with a reciprocal, so neither aggregate transfers between the scales — and that is true of the mean of ratios as much as of the ratio of means, even though the *per-DMU* ratios are exact reciprocals of each other. Both are legitimate statistics for the same hypothesis. Input-oriented the question does not arise and all three packages agree to 1e-13.

The null distribution comes from the smoothed homogeneous bootstrap of [dea_boot](#), driven by the *restricted* scores rather than the variable-returns ones: resampling and smoothing $\{s, 2 - s\}$ and moving each DMU so that its distance to the estimated restricted frontier is the resampled value gives a pseudo-sample whose own frontier satisfies the null by construction. Both technologies are then fitted to that pseudo-sample and the statistic recomputed. Two details of that are choices rather than consequences:

- the *technology* is the pseudo-sample and the points *evaluated* against it are the original observations, as in [dea_boot](#) and as in both reference packages. One thing this buys: in each orientation the untouched half of the data guarantees feasibility, so no replication can be infeasible.

- one pseudo-sample per *replication*, shared by all n evaluated DMUs. The statistic is an average over DMUs, so averaging instead over n independent pseudo-samples would shrink its spread and leave the null distribution too narrow. `npsf::nptestrts` draws a fresh pseudo-sample inside its DMU loop; `rDEA::rts.test` draws one per replication, as here.

The p-value is $(\text{count} + 1)/(B + 1)$ rather than count/B , the standard Monte Carlo correction for the observed statistic being itself a draw under the null. It is therefore never exactly zero, which is the honest report for a quantity whose resolution is $1/(B + 1)$.

Per-DMU tests come free — each replication already yields the whole vector of pseudo-ratios — and are n simultaneous tests, so `scale_efficient` is read at the Sidak level $1 - (1 - \alpha)^{1/n}$ rather than at α . With `h0 = "nirs"` the per-DMU test applied to the scale-inefficient DMUs is what distinguishes decreasing from increasing returns.

Value

An object of class `"dea_rts_test"`, a list whose components include

<code>statistic</code>	The two observed statistics, <code>ratio_of_means</code> and <code>mean_of_ratios</code> , both in $(0, 1]$.
<code>p_value</code> , <code>reject</code>	One-sided lower-tail p-values for each, and whether each rejects at α .
<code>table</code>	One row per DMU: the two scores, the per-DMU <code>scale_eff</code> , its p-value, whether the DMU is scale-efficient at the simultaneous level, and <code>reachable</code> (see below).
<code>boot</code>	The B by 2 matrix of statistics from the null distribution.
<code>boot_ratio</code>	The n by B matrix of per-DMU ratios.
<code>alpha_individual</code>	The Sidak level $1 - (1 - \alpha)^{1/n}$.

How well it works, measured

`dea_sim(returns = 1)` draws a cone, so H_0 is exactly true and the size of the test can be measured rather than assumed. Rejection rate at a nominal 5%, output oriented, $B = 400$, 200 replications (100 at $n = 150$):

```
ab n=60, r=1 ab n=150, r=1 ab n=60, r=0.8 ab n=60, r=0.6
ratio of means ab 0.025 ab 0.000 ab 0.020 ab 0.155
mean of ratios ab 0.050 ab 0.010 ab 0.185 ab 0.780
rDEA ab 0.025 ab 0.000 ab 0.035 ab 0.175
npsf ab 0.315 ab 0.240 ab 0.680 ab 0.990
```

Three things to take from it.

Prefer `mean_of_ratios`. It dominates on this design, 0.780 against 0.155 at $r = 0.6$ with a paired t of -18.2 , at a size no worse. Both are reported because Simar and Wilson give both and because this is one design, but nothing measured recommends the ratio of means.

The size is conservative, not nominal, and the power is low. At $r = 0.8$ the ratio of means rejects 2.0% of the time, below its own size; at $r = 0.6$ — an elasticity of scale of 0.6, which is not subtle — 15.5%. `rDEA::rts.test` sits at 3.5% and 17.5% on the same datasets, so this is the 2002

procedure and not this implementation of it. Do not read a failure to reject as evidence for constant returns at these sample sizes.

`npsf::nptestrts`'s **columns are not comparable**. It rejects a true null 31.5% of the time, because it draws a fresh pseudo-sample inside its DMU loop, so its apparently higher power is the same miscalibration seen from the other side.

What this test cannot be asked

On a bounded input support the input-oriented version of this test is not valid, and no bootstrap repairs it. The variable-returns program cannot put $\sum_i \lambda_i x_{ij}$ below $\min_i x_{ij}$, because a convex combination of the sample's j th input is at least its smallest value. So a DMU whose restricted projection needs $\theta x_{oj} < \min_i x_{ij}$ in any input j has $\theta_{VRS} > \theta_{CRS}$ for a reason that has nothing to do with returns to scale, and no sample size removes it. This is the same identification problem that `dea_sim`'s `x_range` argument exists for, reaching a test instead of a convergence rate. Measured on `dea_sim(returns = 1)`, where the technology *is* a cone and the null is exactly true, so a consistent statistic must approach 1:

n	80	200	800	3200
input, ratio of means	0.887	0.903	0.894	0.898
output, ratio of means	0.970	0.985	0.991	0.997

The input-oriented statistic does not converge, so the input-oriented test rejects a true null with probability tending to one. Splitting the same designs on exactly the condition above separates a consistent statistic from a constant: at $p = 1$ the reachable DMUs give 0.982, 0.991, 0.9991, 0.9995 while the 34% that are not reachable give 0.755, 0.751, 0.771, 0.776.

So reachable is computed per DMU — the output-oriented mirror is $\phi y_{oj} > \max_i y_{ij}$ — reported by `print`, and warned about past a quarter of the sample. The benign case is not "none flagged" but "a share that *shrinks* with n ": output-oriented on the same cone that share runs 0.188, 0.020, 0.050 where input-oriented it holds at 0.362, 0.340, 0.338. Neither `npsf::nptestrts` nor `rDEA::rts.test` reports anything of the kind, and both will run the input-oriented test on such a design without comment.

Separately, Simar and Wilson (2011) show that the full-sample bootstrap used here is not consistent for tests of this kind and propose an m out of n subsampling alternative. That is not implemented; this function is the 2002 procedure, which is what both packages on CRAN provide.

References

- Simar, L. and Wilson, P. W. (1998). Sensitivity analysis of efficiency scores: how to bootstrap in nonparametric frontier models. *Management Science* 44, 49–61.
- Simar, L. and Wilson, P. W. (2002). Non-parametric tests of returns to scale. *European Journal of Operational Research* 139, 115–132.
- Simar, L. and Wilson, P. W. (2011). Inference by the m out of n bootstrap in nonparametric frontier models. *Journal of Productivity Analysis* 36, 33–53.

See Also

`dea_rts` for the per-DMU classification and scale efficiency, `dea_boot` for the resampling scheme this builds on.

Examples

```
sim <- dea_sim(80, p = 2, q = 1, returns = 0.6, seed = 2)
## Output oriented: see 'What this test cannot be asked' above.
r <- dea_rts_test(sim$x, sim$y, orientation = "out", B = 299, seed = 1,
                  progress = FALSE)
r
## The two aggregations are different tests, and on this design they disagree.
head(r$table)
```

dea_sbm

Slacks-based measure of efficiency

Description

Tone's (2001) slacks-based measure: a non-radial efficiency score that is 1 if and only if the DMU is Pareto-Koopmans efficient, units invariant, and monotone in every input and output slack.

Usage

```
dea_sbm(x, y, data = NULL,
        rts = c("vrs", "crs", "nirs", "ndrs"),
        orientation = c("none", "in", "out"),
        peers = TRUE, scaling = TRUE,
        xref = NULL, yref = NULL, dataref = NULL)
```

Arguments

x, y, data	Inputs and outputs, in any of the forms accepted by dea . Every value must be strictly positive: the measure divides each slack by the DMU's own level.
rts	Returns to scale, as in dea . There is no "fdh" option – the measure is defined on a convex technology.
orientation	"none" (the default) scores input and output slack together, as a ratio; "in" scores input slack only and "out" output slack only. All three return a score in (0, 1].
peers, scaling, xref, yref, dataref	As in dea .

Details

The non-oriented measure is

$$\rho = \frac{1 - (1/p) \sum_i s_i^- / x_{io}}{1 + (1/q) \sum_r s_r^+ / y_{ro}}$$

minimized over the technology. It answers a different question from the radial score: not “by what common factor can every input be cut” but “what fraction of each input is being wasted”. The two coincide only when no slack remains at the radial projection, and $\rho \leq \theta$ always.

The fractional program is solved exactly. A ratio of linear forms is not a linear program, and the Charnes-Cooper change of variable used here – introduce $t > 0$, substitute $\Lambda = t\lambda$ and $S = ts$, and normalize the denominator to 1 – turns it into one without approximation.

The oriented measures are one-sided by construction. ρ_I sees only input waste and ρ_O only forgone output, so a DMU can score 1 on one of them while carrying slack on the other side. Use "none" unless the orientation is a substantive assumption about what the DMU controls.

An external reference set can make the program infeasible. The balance constraints are equalities with non-negative slacks, so they require some convex combination of the reference DMUs to weakly dominate the evaluated point in every input *and* every output. A DMU scored against its own sample dominates itself, so this is free; a point scored against someone else's frontier may lie outside the estimated technology, and then there is no solution. `dea_sbm()` reports NA and warns, naming the cause. `dea` has no such problem — its radial score simply falls below 1 for a point outside the technology — so use it when scoring out of sample.

Zeros. Because every slack is divided by the DMU's own level, a zero input or output has no defined contribution. `dea_sbm()` stops rather than substituting a small number, which would make the score depend on the number chosen. Use `dea`, whose radial score tolerates zeros, or `dea_ddf`, whose additive measure never divides by the data.

Value

An object of class "dea", with the same components as `dea`'s return value; `eff` holds ρ and `model` is "sbm". Slacks are always computed, so `efficient` is `eff == 1`.

References

Tone, K. (2001). A slacks-based measure of efficiency in data envelopment analysis. *European Journal of Operational Research* 130, 498–509.

See Also

`dea`, `dea_ddf`

Examples

```
sim <- dea_sim(60, p = 2, q = 1, seed = 1)

rho <- dea_sbm(sim$x, sim$y, rts = "vrs")
rho

## rho <= theta, with equality exactly where the radial projection leaves no
## slack behind.
th <- dea(sim$x, sim$y, rts = "vrs", orientation = "in")
all(rho$eff <= th$eff + 1e-8)
identical(rho$eff == 1, unname(th$efficient))
```

dea_sim	<i>Simulate a technology with known efficiency, and the rate it can be estimated at</i>
---------	---

Description

dea_sim draws a sample from a production technology whose distance functions have a closed form, so that an estimator can be scored against a truth written down before it ran. dea_rate gives the rate at which the DEA estimator can converge on such a design, expressed as the slope of log mean squared error on log sample size.

Usage

```
dea_sim(n, p = 1L, q = 1L, returns = 1, ineff = c("exp", "hnorm"),
        mean_ineff = 0.3, x_range = c(1, 2), seed = NULL,
        frontier = c("cobb", "ces", "facet"), frontier_par = NULL)

dea_rate(p, q, rts = c("vrs", "crs", "nirs", "ndrs", "fdh"),
        what = c("mse", "estimator"))

## S3 method for class 'dea_sim'
print(x, ...)
## S3 method for class 'dea_sim'
plot(x, ...)
```

Arguments

n	Number of DMUs.
p, q	Numbers of inputs and outputs.
returns	Elasticity of scale of the true frontier, in $(0, 1]$. At 1 the technology is a cone, so a constant-returns fit is consistent; below 1 it is strictly concave and a constant-returns fit is inconsistent by construction, which makes it a useful negative control.
ineff	Distribution of the inefficiency term $u \geq 0$: exponential or half-normal. Both are parameterized so that $E[u]$ is mean_ineff, so the two designs are comparable.
mean_ineff	$E[u]$.
x_range	Support of each input, as two increasing positive numbers. Widening it downward matters for any estimand whose projection moves inputs down — see ‘Details’.
frontier	Shape of the true frontier: "cobb" the Cobb-Douglas aggregate (smooth and strictly concave; the default and the original), "ces" a CES aggregate whose curvature frontier_par sets, or "facet" a piecewise-linear frontier with frontier_par facets. All three are homogeneous of degree returns and all agree on the diagonal, $f(c, \dots, c) = c^r$, so returns means the same thing across them. See Frontier geometry below.

frontier_par	The shape's one parameter, or NULL for its default. For "ces" it is $\rho \leq 1$ (default 0.5); for "facet" the number of facets, at least 2 (default 3). Refused for "cobb", which has no free parameter.
seed	Integer seed. The caller's random-number stream is restored afterwards.
rts	Returns-to-scale assumption of the estimator whose rate is wanted.
what	"mse" returns the slope for mean squared error; "estimator" returns it for the estimator itself, which is half as steep.
x	An object of class "dea_sim".
...	Ignored.

Details

The design. Inputs are uniform on x_range^p , by default $[1, 2]^p$. The frontier is the Cobb-Douglas aggregate $f(x) = \prod_j x_j^{r/p}$ with $r = \text{returns}$, and the output set at x is the non-negative part of the ball of radius $f(x)$:

$$T = \{(x, y) : \|y\| \leq f(x)\}.$$

This is a genuine production technology in the sense DEA assumes – f is concave for $r \leq 1$ so T is convex, both f and the norm are monotone so T is freely disposable, and at $r = 1$ it is a cone. Setting $\|y\| = f(x)e^{-u}$ then gives

$$\phi = f(x)/\|y\| = e^u, \quad \theta = e^{-u/r},$$

the second because scaling every input by θ scales f by θ^r . With $q > 1$ the direction of y within the ball is drawn uniformly and independently of u , so the output mix carries no information about efficiency; fixing it instead would put every DMU on one ray, and a q -output problem on a single ray is a one-output problem wearing q columns.

Frontier geometry. A convergence rate is a claim about a class of technologies, and one member of that class cannot confirm it, so the shape of f is an argument. Changing it costs nothing, which is not obvious: both closed forms above survive *any* change of shape provided f stays homogeneous of degree r , because

$$\theta = \min\{t : f(tx) \geq \|y\|\} = \min\{t : t^r f(x) \geq f(x)e^{-u}\} = e^{-u/r},$$

which uses homogeneity and nothing else — not smoothness, not strict concavity, and nothing Cobb-Douglas — while $\phi = f(x)/\|y\|$ holds for any f whatever. So each shape below is built homogeneous of degree r and its truth needs no re-derivation:

"cobb" $f(x) = (\prod_j x_j)^{r/p}$. Smooth and strictly concave.

"ces" $f(x) = ((1/p) \sum_j x_j^\rho)^{r/\rho}$. Nests the others: $\rho \rightarrow 0$ is Cobb-Douglas, $\rho = 1$ is an arithmetic mean and so a flat frontier, and $\rho \rightarrow -\infty$ is Leontief $\min_j x_j$. So ρ moves curvature continuously. $\rho > 1$ is refused rather than clamped: it makes f convex, so T is not convex and a variable-returns estimator converges to the convex hull of T instead of to T . The θ returned would still be the true distance and would no longer be the estimand, so the design would quietly score the estimator against the wrong number.

"facet" $f(x) = (\min_k a'_k x/p)^r$ over frontier_par facets. Piecewise linear, which is the shape a DEA hull can represent exactly, and the shape with flat faces, so the optimal face need not be a point and the projection need not be unique. The normals are *deterministic*: drawn from

the random stream they would move with seed, and “the same geometry at two sample sizes” — which any rate study depends on — would be false. Each is scaled to sum to p , so every facet passes through the diagonal ray and the frontier is a fan of flat faces meeting along it.

At $p = 2$ the scaled normals are collinear, and a minimum of $a'_k x$ over collinear a_k is always attained at an endpoint, so exactly two facets can ever bind however many are asked for; more than two at $p = 2$ is refused for that reason. At $p \geq 3$ interior facets do bind, and the returned facet says which, so a design whose geometry is simpler than it claims can be seen rather than assumed.

theta needs care, and x_range is why it is an argument. theta is the distance to the technology as defined for *all* $x > 0$, and reaching it means scaling inputs down — for an inefficient DMU, below $\min(x)$. A variable-returns hull cannot extrapolate there, so on the default support it does not estimate theta consistently however large the sample: the mean squared error hits a floor and a log-log sweep produces a slope near zero. That is a property of the estimand, not a bug. Three ways out: use phi, since outputs expand at fixed x and stay inside the support; fit $\text{rts} = \text{"crs"}$ against theta on a $\text{returns} = 1$ design, where the technology is a cone and can be scaled without bound; or widen x_range downward so the sample has room beneath the points being scored. The same applies to the directional distance function’s two-sided direction, which also moves inputs down.

The rate. DEA is consistent but not root- n consistent, and the rate worsens with every variable added. For the radial estimator with p inputs and q outputs,

$$|\hat{\theta} - \theta| = O_p(n^{-2/(p+q+1)})$$

under variable returns (Kneip, Park and Simar 1998), $O_p(n^{-2/(p+q)})$ under constant returns (Park, Simar and Weiner 2000), and $O_p(n^{-1/(p+q)})$ for the free disposal hull. Squaring doubles the exponent, so regressing log mean squared error on $\log n$ over a grid of sample sizes has slope $-4/(p+q+1)$, $-4/(p+q)$ and $-2/(p+q)$ respectively. That number, not -1 , is the benchmark: the familiar root- n slope is reached only at $p+q = 3$ under variable returns, and at $p+q = 8$ the slope is $-4/9$, so a hundredfold increase in sample size buys about a factor of 8 in mean squared error where a parametric estimator would buy 100.

These are worst-case rates over a class of technologies. On a frontier that happens to be linear the estimator does better, so a slope *steeper* than the target is not a defect; only a slope materially flatter is.

Value

dea_sim returns an object of class "dea_sim": a list with x and y (matrices), data (both as one data frame), theta and phi (the true input- and output-oriented Farrell efficiencies), u , frontier and design. A $\text{frontier} = \text{"facet"}$ design also carries facet, the index of the facet each DMU sits under, and facet_normals; for the other shapes both are NULL.

dea_rate returns a single negative number.

References

- Kneip, A., Park, B. U. and Simar, L. (1998). A note on the convergence of nonparametric DEA estimators for production efficiency scores. *Econometric Theory* 14, 783–793.
- Park, B. U., Simar, L. and Weiner, C. (2000). The FDH estimator for productivity efficiency scores. *Econometric Theory* 16, 855–877.
- Simar, L. and Wilson, P. W. (2015). Statistical approaches for non-parametric frontier models: a guided tour. *International Statistical Review* 83, 77–110.

See Also[dea](#), [dea_boot](#)**Examples**

```

sim <- dea_sim(200, p = 1, q = 1, returns = 1, seed = 1)
sim

## A constant-returns fit recovers the truth on a constant-returns design.
fit <- dea(sim$x, sim$y, rts = "crs", orientation = "in", slack = FALSE)
c(bias = mean(fit$eff - sim$theta), rmse = sqrt(mean((fit$eff - sim$theta)^2)))

## What the rate ought to be, here and in a bigger problem.
dea_rate(1, 1, "crs")
dea_rate(4, 4, "vrs")

```

dea_subsample

*Confidence intervals by the m out of n bootstrap***Description**

Subsampling intervals for radial efficiency scores: draw $m < n$ observations without replacement, rescale by the estimator's convergence rate, and invert. Simar and Wilson (2011), with the data-driven choice of m following Politis, Romano and Wolf.

Usage

```

dea_subsample(object, m = NULL, B = 2000, alpha = 0.05,
              m_grid = NULL, B_select = 200L, window = 3L,
              seed = NULL, ncores = 1L, progress = interactive())

## S3 method for class 'dea_subsample'
print(x, ...)
## S3 method for class 'dea_subsample'
summary(object, ...)

```

Arguments

object	A self-referenced radial fit from dea . Unlike dea_boot , <code>rts = "fdh"</code> is accepted.
m	Subsample size. NULL chooses it by minimum volatility.
B	Replications at the chosen m.
alpha	Interval level.
m_grid	Candidate values of m. The default is twelve points spaced geometrically between about $n^{0.4}$ and $n^{0.9}$.

B_select	Replications used at each candidate while choosing m . Smaller than B on purpose: the selection only needs the endpoints to be comparable across m .
window	Half-width of the neighbourhood the volatility is computed over.
seed, ncores, progress	As in dea_boot .
x	An object of class "dea_subsample".
...	Ignored.

Details

[dea_boot](#) resamples all n observations. For a boundary estimator that is not known to be consistent: the score's limiting distribution is driven by how many observations land near the frontier, and an n -out-of- n resample reproduces the sample's own boundary rather than the population's. Simar and Wilson (2011) propose subsampling instead.

Writing κ for the convergence exponent that [dea_rate](#) already knows — $2/(p + q + 1)$ under variable returns, $2/(p + q)$ under constant, $1/(p + q)$ under free disposal — $n^\kappa(\hat{\theta}_n - \theta)$ has a limiting distribution, and

$$T_m^* = m^\kappa(\hat{\theta}_m^* - \hat{\theta}_n)$$

estimates it, where $\hat{\theta}_m^*$ scores the same observation against a subsample of size m . Inverting gives the reported interval. No orientation branch is needed: input-oriented θ is biased up and θ_m^* is larger still, output-oriented ϕ is biased down and ϕ_m^* smaller still, and the signs carry themselves.

It also covers the case [dea_boot](#) refuses. Free disposal has no smoothed homogeneous bootstrap — the hull is not convex — and subsampling is the scheme Jeong and Simar (2006) propose for it.

Choosing m is the whole difficulty. The theory asks for $m \rightarrow \infty$ with $m/n \rightarrow 0$ and says nothing about which m to use at the n in front of you. The minimum-volatility rule computes the interval across a grid and takes the m whose neighbourhood moves the endpoints least, on the argument that a flat stretch is where the asymptotics have bitten. Two things about the implementation:

- Only candidates with a *full* window are eligible. A clipped window averages over fewer neighbours, and a standard deviation over fewer points is smaller for no reason connected to stability — so clipping biases the rule toward the extreme m . Observed before this was fixed: on a 100-DMU fit the rule picked the first grid point at volatility 0.0107 against 0.0109 for a candidate in the middle of a genuinely flat stretch, purely because one was scored on four neighbours and the other on seven.
- Politis, Romano and Wolf choose m for *one* parameter; here there are n intervals at once. The volatility is therefore computed on the *mean* endpoints across DMUs and one m serves all of them. That aggregation is a choice the method does not make, so `m_grid`, `endpoint_path` and `volatility` are returned for inspection and `summary()` prints the path.

Value

An object of class "dea_subsample" whose table has one row per DMU with `eff`, `bias_corrected`, `ci_lower`, `ci_upper` and `n_valid`. Also `m`, `m_grid`, `endpoint_path`, `volatility`, `eligible` and `kappa`.

What subsampling costs, and who pays

Scoring an original observation against a subsample can have no solution at all: under variable returns the point may lie outside the smaller hull, and then there is no answer rather than a large one. That is not an edge case, and the burden falls on exactly the DMUs an analyst cares about. Share of replications with no solution, on `dea_sim(returns = 0.9)`, output-oriented, variable returns:

	$m = n^{0.4}$	$m = n^{0.6}$	$m = n^{0.8}$
$n = 60$, all DMUs	31.3%	15.7%	6.3%
$n = 60$, called efficient	58.1%	39.2%	22.1%
$n = 60$, interior	26.5%	11.5%	3.5%
$n = 150$, all DMUs	27.3%	13.3%	5.0%
$n = 150$, called efficient	59.6%	42.7%	24.9%

The worst single DMU lost between 63% and 97% of its draws. So the interval for a frontier DMU is built from the fewest observations, and a smaller m makes that worse at exactly the rate that makes the asymptotics better.

`n_valid` is therefore reported per DMU, an interval needs at least $\lceil 2/\alpha \rceil$ valid draws to be reported at all — below that the $\alpha/2$ quantile is extrapolation into an empty tail rather than an order statistic — and anything short comes back NA with a warning naming the count.

Does it cover better than `dea_boot`

Measured against `dea_sim`'s closed-form truth, 100 replications, output-oriented variable returns, nominal 95%, both methods on the same fit in every replication:

	coverage	frontier	interior	width
$n = 50$, <code>dea_boot</code>	0.814	0.794	0.819	0.170
$n = 50$, <code>dea_subsample</code>	0.811	0.602	0.871	0.157
$n = 200$, <code>dea_boot</code>	0.879	0.710	0.897	0.089
$n = 200$, <code>dea_subsample</code>	0.925	0.695	0.950	0.097

Three things to take from it, and only one of them is a recommendation.

At $n = 200$ it is a real improvement, and it is the interior that improves. Overall coverage rises 0.046 with a paired standard error of 0.0047 ($t = 9.8$), and interior coverage reaches 0.950 against a nominal 0.95 — which is the package's largest documented inference weakness closed, not narrowed.

At $n = 50$ it is not an improvement at all. The paired difference is -0.003 and does not resolve. Subsampling needs sample size, which is what $m \rightarrow \infty$ with $m/n \rightarrow 0$ asks for and what $n = 50$ cannot give.

It never helps the frontier DMUs — 0.794 to 0.602 at $n = 50$ and 0.710 to 0.695 at $n = 200$ — because that is where the infeasibility above concentrates. If the units of interest are the ones the fit calls efficient, this method has nothing to offer them and the smaller m the selection prefers makes it worse.

References

- Jeong, S.-O. and Simar, L. (2006). Linearly interpolated FDH estimators of multivariate efficiency scores. *Journal of Multivariate Analysis* 97, 2141–2161.
- Politis, D. N., Romano, J. P. and Wolf, M. (1999). *Subsampling*. Springer.
- Simar, L. and Wilson, P. W. (2011). Inference by the m out of n bootstrap in nonparametric frontier models. *Journal of Productivity Analysis* 36, 33–53.

See Also

[dea_boot](#) for the full-sample bootstrap this is an alternative to, [dea_rate](#) for the exponent, [dea](#).

Examples

```
sim <- dea_sim(80, p = 2, q = 1, returns = 0.9, seed = 5)
fit <- dea(sim$x, sim$y, rts = "vrs", orientation = "out", slack = FALSE)

r <- dea_subsample(fit, m = 20, B = 200, seed = 1, progress = FALSE)
r

## The m selection, as a path to be inspected rather than trusted.

sel <- dea_subsample(fit, B = 200, B_select = 50, seed = 1, progress = FALSE)
summary(sel)
```

dea_undesirable	<i>Efficiency with undesirable outputs</i>
-----------------	--

Description

Radial efficiency when some outputs are bad — pollution, defects, non-performing loans, mortality — by either the Seiford and Zhu (2002) transformation or by treating the bad outputs as inputs. Both reduce to [dea](#) on rearranged data; what this function adds is the refusals, the diagnostics and a slack that reads directly as reducible bad output.

Usage

```
dea_undesirable(x, y, bad, data = NULL,
               method = c("translate", "input"),
               rts = c("vrs", "crs", "nirs", "ndrs"),
               orientation = c("in", "out"), trans = NULL, ...)

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

Arguments

x, y, data	Inputs and the <i>desirable</i> outputs, in any of the forms accepted by dea . For the print method, an object of class "dea_undesirable".
bad	The undesirable outputs, in any of the same forms. One column per bad output, one row per DMU.
method	"translate" for Seiford and Zhu (2002), "input" for Hailu and Veeman (2001). There is no default that is right for every question; see Which route below.
rts, orientation	As in dea . method = "translate" requires "vrs" and "in" and refuses anything else, for the reason given below.
trans	The translation constant for method = "translate": one value per bad output or a single value for all of them. The default is $\max(\text{bad}) + 1$ per column, which is deaR's. The score does not depend on it.
...	Passed to dea — slack, peers, scaling, multipliers, xref/yref.

Value

An object of class `c("dea_undesirable", "dea")`, so every method for "dea" applies. In addition to the usual components:

bad	The undesirable outputs as supplied.
bad_method, bad_trans	The route used, and the translation constants (NULL for method = "input").
slack_bad	One column per bad output: the amount by which it could be reduced, in the caller's own units, under either route.

Which route

A bad output is not an output. Free disposal runs the wrong way for it, so listing it in y and solving the ordinary program asks the technology to let a unit produce *more* of it and rewards a unit for doing so. Three routes answer this; two need no new solver and are offered here.

method = "translate" replaces each bad b by $w - b$, so that more of the transformed variable is better, and runs the ordinary program. The transformed variable is a genuine output: a cleaner unit has more of it, and free disposal of it correctly says a unit could always pollute more.

method = "input" puts the bad among the inputs. Free disposal then runs the right way, it needs no constant and no invariance, and it works under every technology and orientation. What it does not encode is **null-jointness**: the program is free to recommend less bad at no cost in good output, which the physics of most production forbids. That objection is real, and it is why the third route exists.

The third route, the weak-disposability technology of Fare and Grosskopf, puts the bads on equality rows with an abatement factor and is the only one of the three that encodes null-jointness. It needs a new program rather than a rearrangement of this one, and is [dea_weak](#).

Why the translate route refuses most models

w is arbitrary, so the answer is only meaningful where the model does not depend on it — that is, where the model is invariant to translating that output. Measured on a 40-DMU design, running this transformation at $w = \max(b) + 1, 20, 100$ and 1000, the largest change in *any* DMU's score:

	rts	input	output
vrs		7.0e-11	2.5e-01
crs		1.8e-02	2.6e-02
nirs		4.0e-03	2.5e-01
ndrs		1.8e-02	2.6e-02

It holds in exactly one cell, with eight orders of magnitude between that cell and every other — not a tolerance question. And not "a convex technology" either: `nirs` is convex and moves by 4e-3. So the other combinations are refused rather than served with a number, as `dea_profit` refuses a cone. `deaR::undesirable_basic`'s help says `rts` must be `vrs` and says nothing about the orientation, which is the half of the precondition that bites hardest: studying outputs is exactly when a reader reaches for an output orientation.

The second precondition is numerical, and the failure is silent. A large w makes the transformed column nearly constant, and after it is scaled to mean one its values differ only in the last digits. Writing `rel` for the column's relative spread $\text{range}(b)/\text{mean}(w - b)$, measured against the default on the same design:

rel	largest score change	DMUs unsolved
1.4e+00 (default)	—	0
7.7e-04	1.7e-08	0
7.7e-06	2.5e-06	1
7.7e-07	0.188	3
7.7e-09	0.188	0

The last row is why this is warned about rather than merely documented: every program reports a clean optimum, nothing is NA, and the scores are wrong by 0.188. A relative spread below 1e-3 warns. The default never triggers it.

References

Hailu, A. and Veeman, T. S. (2001). Non-parametric productivity analysis with undesirable outputs: an application to the Canadian pulp and paper industry. *American Journal of Agricultural Economics* 83, 605–616.

Seiford, L. M. and Zhu, J. (2002). Modeling undesirable factors in efficiency evaluation. *European Journal of Operational Research* 142, 16–20.

Fare, R. and Grosskopf, S. (2004). Modeling undesirable factors in efficiency evaluation: comment. *European Journal of Operational Research* 157, 242–245.

See Also

`dea_weak` for the weak-disposability technology, `dea`, `dea_ddf` for a directional measure that takes an arbitrary direction.

Examples

```
set.seed(4)
n <- 40
x <- matrix(runif(n * 2, 2, 10), n, 2, dimnames = list(NULL, c("x1", "x2")))
good <- matrix((x[, 1] * x[, 2])^0.4 * exp(-abs(rnorm(n, 0, 0.2)))), n, 1,
               dimnames = list(NULL, "output"))
co2 <- matrix(good[, 1]^1.2 * exp(rnorm(n, 0, 0.15)), n, 1,
               dimnames = list(NULL, "CO2"))

r <- dea_undesirable(x, good, co2)
r
head(r$slack_bad)      # reducible CO2, in the caller's units

## The same data with the bad treated as an input, which needs no constant.
dea_undesirable(x, good, co2, method = "input")

## The score does not depend on the translation constant -- that is the whole
## justification for the route, and it is why the other models are refused.
all.equal(dea_undesirable(x, good, co2, trans = 1000)$eff, r$eff,
          tolerance = 1e-8)
```

dea_weak

Weak disposability of undesirable outputs

Description

The directional distance function on the weak-disposability technology, where bad outputs can be reduced only by scaling the activity back rather than thrown away freely. Chung, Fare and Grosskopf (1997), with the variable-returns formulation of Kuosmanen (2005).

Usage

```
dea_weak(x, y, bad, data = NULL,
         direction = c("goods_bads", "all", "bads"),
         rts = c("vrs", "crs"),
         technology = c("kuosmanen", "fare"),
         peers = TRUE, scaling = TRUE)

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

Arguments

x, y, data	Inputs and the <i>good</i> outputs, in any of the forms accepted by dea . For the print method, an object of class "dea_weak".
bad	The undesirable outputs, one column each.

direction	"goods_bads" raises good output and lowers bad output together, holding inputs — the Chung-Fare-Grosskopf default. "all" also contracts inputs; "bads" moves only the bads. A numeric vector of length $p + q + b$ or a matrix with one row per DMU gives an explicit direction. Every entry is a magnitude : the signs are already in the program, so a negative entry is refused.
rts	"vrs" or "crs". The one-sided technologies have no standard weak-disposability form and are not offered.
technology	Which variable-returns formulation. Ignored under "crs", where the two are the same program.
peers, scaling	As in dea_ddf .
...	Ignored.

Details

[dea_undesirable](#) offers two routes that are honest rearrangements of an ordinary program, and neither encodes the one thing everybody believes about pollution: that you cannot simply stop producing it. Under free disposal a DMU may always cut its bad output at no cost, so the model will cheerfully prescribe less pollution and the same product.

Weak disposability is an equality row:

$$\sum_j \lambda_j x_{ij} \leq x_{io} - \beta g_{xi}$$

$$\sum_j \lambda_j y_{rj} \geq y_{ro} + \beta g_{yr}$$

$$\sum_j \lambda_j b_{fj} = b_{fo} - \beta g_{bf}$$

so the bads can be reduced only by moving along the technology rather than by assumption.

Null-jointness is a property of the data, not a constraint. The technology means what it is supposed to mean only if no good output can be had without some bad; a DMU with positive goods and zero bads says the opposite, and one such row lets the frontier claim that clean production is already feasible. It cannot be imposed by a constraint, so it is checked, reported by `print()`, and warned about.

Value

An object of class `c("dea_weak", "dea")`. `eff` is β : the proportion by which good output could rise and bad output fall at the same time, 0 on the frontier and larger worse, as in [dea_ddf](#). Also `lambda` and, for the Kuosmanen form, `mu`; `bad`; and `null_joint` with `n_clean`.

Which variable-returns formulation, and how much it matters

Fare and Grosskopf's variable-returns version applies one abatement factor to the whole sample, which in linear form is the single-intensity program above with $\sum \lambda = 1$. Kuosmanen (2005)

showed this is not the minimal technology satisfying weak disposability and variable returns, and gave a two-vector form in which each DMU carries its own abatement factor:

$$\sum_j (\lambda_j + \mu_j) x_{ij} \leq x_{io} - \beta g_{xi}, \quad \sum_j (\lambda_j + \mu_j) = 1$$

with the good and bad rows still in λ alone, so $\sum \lambda$ may fall below 1 while the input side still averages to one. That is strictly more freedom — $\mu = 0$ is always available — so Kuosmanen's technology contains Fare and Grosskopf's and $\beta_K \geq \beta_{FG}$ is an identity.

How much larger is not an identity, and the answer depends almost entirely on the *direction*, which neither paper discusses. From `horserace/weak_disposability_experiment.R`, 50 replications at $n = 60$:

direction	coupling	μ used	mean gap	max gap
goods_bads	prop	6.6%	0.00091	0.113
goods_bads	steep	0.7%	0.00017	0.107
bads	prop	16.6%	0.00435	0.203
bads	flat	30.0%	0.01347	0.393
all	flat	2.1%	0.00039	0.097

On the standard direction the second vector is used for a few per cent of DMUs and the mean gap is a thousandth — but the largest gap is 0.1 even there, so the choice bites a few units hard rather than everyone a little. It matters most when the direction asks for bads to be cut and nothing else, which is exactly when scaling the activity back is the attractive move and μ is what permits it.

What it does not much change is *who* is on the frontier: the efficient set differs between the formulations for 0.2% to 0.7% of DMUs. The formulation is a question about the size of measured inefficiency far more than about its classification.

Kuosmanen's is the default because it is the one satisfying minimal extrapolation. Fare and Grosskopf's is kept for reproducing published results, the same reason `dea_mult` keeps the 1982 model. Under "crs" there is no convexity row, μ only adds slack that free disposal already allows, and the two are the same program — technology is then ignored and says so.

Does weak disposability change the answer

If it did not, the honest thing would be to say so and point at the one-liner in `dea_undesirable`. Against the same direction and the same data, with the bad treated as a freely disposable input instead:

coupling	mean weak	mean strong	difference	rank correlation
prop	0.0508	0.0519	-0.0011	0.955
steep	0.0663	0.0697	-0.0034	0.951
flat	0.0866	0.0950	-0.0084	0.917

Weak disposability gives the smaller β — it is the smaller technology, so less improvement is available — by between 2% and 10% of the level, and the rankings agree at 0.92 to 0.96 rather than exactly.

The gap is largest where the good-bad coupling is *weakest*, which is the opposite of the natural guess. When bads move almost independently of output, the free-disposal model finds plenty of

apparently costless abatement — peers with similar output and much lower emissions — and weak disposability forbids exactly that. When bads are tightly tied to output there is little free abatement to find and the two models nearly agree.

References

- Chung, Y. H., Fare, R. and Grosskopf, S. (1997). Productivity and undesirable outputs: a directional distance function approach. *Journal of Environmental Management* 51, 229–240.
- Fare, R., Grosskopf, S., Lovell, C. A. K. and Pasurka, C. (1989). Multilateral productivity comparisons when some outputs are undesirable. *Review of Economics and Statistics* 71, 90–98.
- Kuosmanen, T. (2005). Weak disposability in nonparametric production analysis with undesirable outputs. *American Journal of Agricultural Economics* 87, 1077–1082.

See Also

[dea_undesirable](#) for the two cheaper routes, [dea_ddf](#) for the directional model on an ordinary technology.

Examples

```
set.seed(7)
n <- 50
x <- matrix(runif(n * 2, 5, 20), n, 2,
            dimnames = list(NULL, c("capital", "energy")))
product <- matrix((x[, 1] * x[, 2])^0.4 * exp(-abs(rnorm(n, 0, 0.2)))), n, 1,
                dimnames = list(NULL, "product"))
## Null-joint by construction: emissions rise with output, so nobody is clean.
co2 <- matrix(product[, 1]^1.1 * exp(rnorm(n, 0, 0.15)), n, 1,
            dimnames = list(NULL, "CO2"))

w <- dea_weak(x, product, co2)
w

## The bad-output row is an equality, which is the whole model.
peer <- as.numeric(w$lambda %*% co2)
max(abs(peer - (co2[, 1] - w$eff * co2[, 1])))
```

dea_wr

Weight restrictions, assurance regions and the cone-ratio model

Description

An unrestricted DEA score lets every DMU pick the prices that flatter it most, including prices that value an input or an output at zero. `dea_wr()` solves the radial model with the admissible weights restricted: bounds on individual multipliers (Dyson and Thanassoulis 1988), bounds on their ratios — assurance regions (Thompson et al. 1986) — and general linear inequalities (Wong and Beasley 1990). `dea_cone()` is the cone-ratio model of Charnes et al. (1989), which needs no new linear program at all.

Usage

```

wr_bound(on = c("v", "u"), index, lower = NA, upper = NA)
wr_ratio(on = c("v", "u"), numerator, denominator, lower = NA, upper = NA)
wr_linear(v = NULL, u = NULL, type = c("<=", ">=", "="), rhs = 0)

dea_wr(x, y, restrictions, data = NULL,
       rts = c("vrs", "crs", "nirs", "ndrs"),
       orientation = c("in", "out"),
       scaling = TRUE,
       xref = NULL, yref = NULL, dataref = NULL)

dea_cone(x, y, A = NULL, B = NULL, data = NULL,
         rts = c("crs", "vrs", "nirs", "ndrs"),
         orientation = c("in", "out"), ...)
```

Arguments

on	Which family of weights the restriction is about: "v" for the inputs, "u" for the outputs.
index, numerator, denominator	Column positions, counted within that family.
lower, upper	Bounds. Give one or both.
v, u	For wr_linear(), coefficient vectors on the input and the output weights, one entry per column; either may be NULL.
type, rhs	The sense and right-hand side of the inequality.
x, y, data, rts, orientation, scaling, xref, yref, dataref	As in dea .
restrictions	A restriction set built by the three constructors above and combined with c(), or NULL for none.
A, B	Generator matrices for the admissible input and output weight cones, one row per generator and one column per input or output. NULL means the identity, which is no restriction.
...	Passed to dea .

Details

The score comes from the multiplier program, and with restrictions the two forms are no longer interchangeable. Strong duality still holds, but the dual of the restricted multiplier program is not the ordinary envelopment program. It acquires one extra non-negative term per restriction, entering the constraints in exactly the place a slack does:

$$X'\lambda + A_x z \leq \theta x_o, \quad Y'\lambda - A_y z \geq y_o, \quad z \geq 0.$$

Those Az terms are the *residue*. They are not slacks and the λ beside them does not project the DMU onto a point of the technology. So dea_wr() reports the score and the weights and nothing

else, and `peers()` and `slacks()` on the result raise an error explaining why rather than returning something plausible. That is the reason this is a separate function and not an argument to `dea`.

With `restrictions = NULL` the two forms can still be compared, and they agree to solver precision; the package's tests assert it, because it is what makes the restricted results believable.

A restriction can only make a DMU look worse. Shrinking the admissible weight set shrinks the feasible region of a maximum, so the input-oriented score falls; the output-oriented program is a minimum, so its score rises. Either way the DMU loses ground. A restriction that improved a score would mean a row written with the wrong sign.

Restrictions are written in the caller's units. The solver divides every column by its mean, so the weight it reports applies to x_i/s_i and the caller's weight is v_i/s_i ; the coefficients of a restriction are divided by the same factors on the way in. Getting that wrong would not error, it would quietly impose a different restriction, so the tests require the restricted score to be identical with scaling on and off.

Restrictions can make the program infeasible, and that is the one place a DEA program here can be infeasible on ordinary data — ask for weights no price vector satisfies and there is no answer. Those DMUs come back NA with a warning that points at the restrictions rather than at the data, because an unrestricted fit of the same data cannot fail that way.

The cone-ratio model needs no new program. When the admissible weights are polyhedral cones in *sum* form, $V = \{A'\alpha : \alpha \geq 0\}$ and $U = \{B'\gamma : \gamma \geq 0\}$, substituting $v = A'\alpha$ into $v'x$ gives $\alpha'(Ax)$, so the cone-ratio model is exactly a CCR model on transformed data $\tilde{X} = XA'$, $\tilde{Y} = YB'$ — and $\alpha \geq 0$ is what an ordinary DEA program already imposes.

`dea_cone()` and `dea_wr()` are *not* wrappers for each other. A cone in *sum* form becomes a set of inequalities only through its polar, which is a separate computation and is not generally as small. Where the two can be written both ways they agree, which the package's tests check on a case where the restriction binds.

Value

`dea_wr()` returns an object of class `c("dea_wr", "dea")` carrying `eff`, the weights `v`, `u` and `u0`, and `status`. **It deliberately carries no `lambda`, `peers` or `slacks`**; see **Details**. `dea_cone()` returns an ordinary "dea" object, because it is an ordinary fit on transformed data, with the generators recorded in `$cone`.

References

- Charnes, A., Cooper, W. W., Wei, Q. L. and Huang, Z. M. (1989). Cone ratio data envelopment analysis and multi-objective programming. *International Journal of Systems Science* 20, 1099–1118.
- Dyson, R. G. and Thanassoulis, E. (1988). Reducing weight flexibility in data envelopment analysis. *Journal of the Operational Research Society* 39, 563–576.
- Thompson, R. G., Singleton, F. D., Thrall, R. M. and Smith, B. A. (1986). Comparative site evaluations for locating a high-energy physics lab in Texas. *Interfaces* 16, 35–49.
- Wong, Y.-H. B. and Beasley, J. E. (1990). Restricting weight flexibility in data envelopment analysis. *Journal of the Operational Research Society* 41, 829–835.

See Also[dea](#), [multipliers](#), [dea_cross](#)**Examples**

```
sim <- dea_sim(40, p = 2, q = 2, seed = 3)

## An assurance region: neither input may be valued more than twice the other.
rs <- c(wr_ratio("v", 2, 1, lower = 0.5, upper = 2),
        wr_ratio("u", 2, 1, lower = 0.5, upper = 2))
rs
fit <- dea_wr(sim$x, sim$y, rs, rts = "crs")
fit

## Restricting the weights can only cost a DMU ground.
free <- dea(sim$x, sim$y, rts = "crs", slack = FALSE)
all(fit$eff <= free$eff + 1e-9)

## A floor under one weight, so no input can be valued at nothing.
dea_wr(sim$x, sim$y, wr_bound("v", 1, lower = 0.05), rts = "crs")

## The cone-ratio model, as a transformation of the data.
A <- rbind(c(1, 0.4), c(0.3, 1))
dea_cone(sim$x, sim$y, A = A, rts = "crs")
```

Index

- * **datasets**
 - charnes1981, [5](#)
- * **package**
 - DEA-package, [3](#)
- bw.nrd0, [15](#)
- bw.SJ, [15](#)
- bw.ucv, [15](#)
- c.dea_restriction (dea_wr), [60](#)
- charnes1981, [5](#), [24](#)
- dea, [3](#), [7](#), [10](#), [12–16](#), [18–32](#), [34](#), [35](#), [37](#), [39–43](#), [46](#), [47](#), [51](#), [54–57](#), [61–63](#)
- dea-methods, [10](#)
- DEA-package, [3](#)
- dea_add, [3](#), [12](#), [32](#), [34](#)
- dea_boot, [4](#), [9](#), [10](#), [15](#), [33](#), [34](#), [38](#), [39](#), [41–43](#), [45](#), [51](#), [52](#), [54](#)
- dea_categorical, [3](#), [19](#), [37](#)
- dea_cone, [4](#)
- dea_cone (dea_wr), [60](#)
- dea_cost, [4](#), [21](#)
- dea_cross, [4](#), [12](#), [23](#), [24](#), [63](#)
- dea_ddf, [3](#), [10](#), [12](#), [14](#), [22](#), [23](#), [26](#), [47](#), [56](#), [58](#), [60](#)
- dea_malmquist, [4](#), [29](#)
- dea_mult, [3](#), [31](#), [59](#)
- dea_nd, [3](#), [20](#), [35](#)
- dea_panel, [3](#)
- dea_panel (dea_malmquist), [29](#)
- dea_profit, [4](#), [27](#), [56](#)
- dea_profit (dea_cost), [21](#)
- dea_rate, [4](#), [6](#), [9](#), [18](#), [52](#), [54](#)
- dea_rate (dea_sim), [48](#)
- dea_reg, [4](#), [37](#)
- dea_revenue, [4](#)
- dea_revenue (dea_cost), [21](#)
- dea_rts, [3](#), [10](#), [31](#), [40](#), [42](#), [43](#), [45](#)
- dea_rts_test, [4](#), [41](#), [42](#)
- dea_sbm, [3](#), [8](#), [10](#), [12](#), [14](#), [22](#), [28](#), [46](#)
- dea_sim, [4](#), [10](#), [17](#), [45](#), [48](#)
- dea_subsample, [4](#), [18](#), [51](#)
- dea_undesirable, [3](#), [33](#), [54](#), [58–60](#)
- dea_weak, [3](#), [55](#), [56](#), [57](#)
- dea_wr, [4](#), [60](#)
- efficiency, [8](#), [10](#), [13](#), [27](#)
- efficiency (dea-methods), [10](#)
- efficiency.dea_cross (dea_cross), [24](#)
- efficiency.dea_price (dea_cost), [21](#)
- fitted.dea (dea-methods), [10](#)
- fitted.dea_price (dea_cost), [21](#)
- multipliers, [4](#), [8](#), [25](#), [63](#)
- multipliers (dea-methods), [10](#)
- nobs.dea (dea-methods), [10](#)
- nobs.dea_cross (dea_cross), [24](#)
- nobs.dea_price (dea_cost), [21](#)
- peers (dea-methods), [10](#)
- peers.dea_price (dea_cost), [21](#)
- plot.dea (dea-methods), [10](#)
- plot.dea_boot (dea_boot), [15](#)
- plot.dea_cross (dea_cross), [24](#)
- plot.dea_price (dea_cost), [21](#)
- plot.dea_rts (dea_rts), [40](#)
- plot.dea_sim (dea_sim), [48](#)
- print.dea (dea-methods), [10](#)
- print.dea_boot (dea_boot), [15](#)
- print.dea_categorical (dea_categorical), [19](#)
- print.dea_cross (dea_cross), [24](#)
- print.dea_malmquist (dea_malmquist), [29](#)
- print.dea_mult (dea_mult), [31](#)
- print.dea_nd (dea_nd), [35](#)
- print.dea_panel (dea_malmquist), [29](#)
- print.dea_price (dea_cost), [21](#)
- print.dea_reg (dea_reg), [37](#)

`print.dea_restriction (dea_wr)`, 60
`print.dea_rts (dea_rts)`, 40
`print.dea_rts_test (dea_rts_test)`, 42
`print.dea_sim (dea_sim)`, 48
`print.dea_subsample (dea_subsample)`, 51
`print.dea_undesirable`
 (`dea_undesirable`), 54
`print.dea_weak (dea_weak)`, 57
`print.dea_wr (dea_wr)`, 60

`slacks (dea-methods)`, 10
`summary.dea (dea-methods)`, 10
`summary.dea_boot (dea_boot)`, 15
`summary.dea_categorical`
 (`dea_categorical`), 19
`summary.dea_cross (dea_cross)`, 24
`summary.dea_malmquist (dea_malmquist)`,
 29
`summary.dea_mult (dea_mult)`, 31
`summary.dea_panel (dea_malmquist)`, 29
`summary.dea_price (dea_cost)`, 21
`summary.dea_reg (dea_reg)`, 37
`summary.dea_rts (dea_rts)`, 40
`summary.dea_rts_test (dea_rts_test)`, 42
`summary.dea_subsample (dea_subsample)`,
 51

`wr_bound (dea_wr)`, 60
`wr_linear (dea_wr)`, 60
`wr_ratio (dea_wr)`, 60