Package: additiveDEA (via r-universe)

October 21, 2024

Type Package

Title Additive Data Envelopment Analysis Models

Version 1.1

Date 2017-10-02

Author Andreas Diomedes Soteriades

Maintainer Andreas Diomedes Soteriades <andreassot10@yahoo.com>

Description Provides functions for calculating efficiency with two types of additive Data Envelopment Analysis models: (i) Generalized Efficiency Measures: unweighted additive model (Cooper et al., 2007 <doi:10.1007/978-0-387-45283-8>), Range Adjusted Measure (Cooper et al., 1999, <doi:10.1023/A:1007701304281>), Bounded Adjusted Measure (Cooper et al., 2011 <doi:10.1007/s11123-010-0190-2>), Measure of Inefficiency Proportions (Cooper et al., 1999 <doi:10.1023/A:1007701304281>), and the Lovell-Pastor Measure (Lovell and Pastor, 1995 <doi:10.1016/0167-6377(95)00044-5>); and (ii) the Slacks-Based Measure (Tone, 2001 <doi:10.1016/S0377-2217(99)00407-5>). The functions provide several options: (i) constant and variable returns to scale; (ii) fixed (non-controllable) inputs and/or outputs; (iii) bounding the slacks so that unrealistically large slack values are avoided; and (iv) calculating the efficiency of specific Decision-Making Units (DMUs), rather than of the whole sample. Package additiveDEA also provides a function for reducing computation time when datasets are large.

License GPL-2

Depends R (>= 3.1.0), lpSolveAPI

Imports Benchmarking

URL https://www.r-project.org

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-10-02 11:14:36 UTC

Contents

`dea.fast` *Faster Solving of Additive DEA Models with Large Datasets*

Description

Reduce calculation time of additive DEA efficiency models when the data comprise of several thousand DMUs

Usage

```
dea.fast(base, noutput, fixed = NULL, rts = 2, bound = NULL,
  add.model = c("additive", "RAM", "BAM", "MIP", "LovPast", "SBM"),
  blockSize = 200)
```

Arguments

`base`	A data frame with N rows and S+M columns, where N is the number of Decision-Making Units (DMUs), S is the number of outputs and M is the number of inputs.
`noutput`	The number of outputs produced by the DMUs. All DMUs must produce the same number of outputs.
`fixed`	A numeric vector containing column indices for fixed (non-controllable) outputs and/or inputs (if any) in the data. Defaults to NULL.
`rts`	Returns to scale specification. 1 for constant returns to scale and 2 (default) for variable returns to scale.
`bound`	A data frame with N rows and S+M columns containing user-defined bounds on the slacks of each DMU. If bounds are supplied by the user in cases where some outputs and/or inputs are fixed, values should be 0 for these fixed variables. Same for slacks that do not require bounds. Defaults to NULL.
`add.model`	Additive model to calculate efficiency. additive: unweighted additive model (Cooper et al., 2007); RAM: Range Adjusted Measure (Cooper et al., 1999; 2001); BAM: Bounded Adjusted Measure (Cooper et al., 2011); MIP: Measure of Inefficiency Proportions (Cooper et al., 1999); LovPast: the Lovell-Pastor Measure (Lovell and Pastor, 1995); SBM: Slacks-Based Measure (Tone, 2001).
`blockSize`	How many DMUs should each sub-problem comprise of? Defaults to 200.

Details

dea.fast speeds up computation time of functions dea.gem and dea.sbm when the data comprise of several thousand DMUs. It does so by dividing the data into several blocks consisting of a few hundred DMUs. Then, it finds the efficient DMUs in each block. The next step is to merge the efficient DMUs into one final set and to find the efficient DMUs in this set. Finally, the DMUs in each block are benchmarked against the DMUs that were found to be efficient in the final set of the previous step. See Newsletter 16 in http://www.saitech-inc.com/Products/Prod-DSP.asp. If N is not divisible by blockSize, dea.fast will split the data into a number of even blocks plus a final block with the remaining DMUs. For instance, if N=1050 and blockSize=200, there will be five blocks with 200 DMUs and a sixth one with 50 DMUs.

Value

Returns a numeric vector containing the (in)efficiency scores of the DMUs.

Note

The presence of DMUs with solution status other than 0 (see dea.gem and dea.sbm) will result in dea.fast NOT working. Ensure that there is a solution for all DMUs or, when the solution status is 5, that the data are scaled appropriately.

Extreme care is needed when add.model = 'RAM' when ranges are too large relative to the slacks. In such a case, the slack-range ratios can be so small that an inefficient DMU may seem to have near-zero inefficiency (see Cooper et al., 1999). This makes it extremely hard for the algorithm to distinguish between efficient and inefficient DMUs when the former must be separated from the latter within each block (as described earlier). Avoiding using RAM with dea.fast for the time being is strongly recommended.

Author(s)

Andreas Diomedes Soteriades, <andreassot10@yahoo.com>

References

Cooper W. W., Park K. S., 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

Cooper W. W., Park K. S., Pastor J. T. (2001) The range adjusted measure (RAM) in DEA: a response to the comment by Steinmann and Zweifel. *Journal of Productivity Analysis*, **15**, 145–152

Cooper W. W., Pastor J. T., Borras F., Aparicio J., Pastor D. (2011) BAM: a bounded adjusted measure of efficiency for use with bounded additive models. *Journal of Productivity Analysis*, **35**, 85–94

Cooper W. W., Seiford L., Tone K. (2007) *Data Envelopment Analysis: a comprehensive text with models, applications, references and DEA-Solver software*. New York: Springer

Lovell, C. A. K., Pastor J. T. (1995) Units invariant and translation invariant DEA models. *Operations Research Letters*, **18**, 147–151

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

See Also

dea.gem, dea.sbm

Examples

```
# Get data from package Benchmarking:
library(Benchmarking)
data(pigdata)
base <- pigdata[, 2:9][, c(7,8,1:6)]
# Create trivial but large dataset
base <- rbind(base,base,base,base)
system.time(dea.fast(base, noutput= 2, rts= 2,
  add.model= "LovPast", blockSize = 200))

## The function is currently defined as
function (base, noutput, fixed = NULL, rts = 2, bound = NULL,
    add.model = c("additive", "RAM", "BAM", "MIP", "LovPast",
        "SBM"), blockSize = 200)
{
    baseEfficient <- list()
    n <- nrow(base)
    mod <- (n - (n%%blockSize))/blockSize
    blocks <- c(1, 1:mod * blockSize + 1)
    for (i in 1:mod) {
        aux <- blocks[i]:(blocks[i + 1] - 1)
        base1 <- base[aux, ]
        bound1 <- bound[aux, ]
        if (add.model != "SBM") {
            eff <- round(dea.gem(base = base1, noutput, fixed,
                rts, bound = bound1, add.model)$eff, 7)
            index <- which(is.na(eff))
            if (length(index) > 0) {
                eff[index] <- round(dea.gem(base = base1, noutput,
                  fixed, rts, bound = bound1, add.model, whichDMUs = index)$eff,
                  7)
            }
            baseEfficient[[i]] <- base1[which(eff == 0), ]
        }
        else {
            eff <- round(dea.sbm(base = base1, noutput, fixed,
                rts, bound = bound1)$eff, 7)
            index <- which(is.na(eff))
            if (length(index) > 0) {
                eff[index] <- round(dea.sbm(base = base1, noutput,
                  fixed, rts, bound = bound1, whichDMUs = index)$eff,
                  7)
            }
            baseEfficient[[i]] <- base1[which(eff == 1), ]
        }
    }
    if (n%%blockSize != 0) {
        aux <- (n - (n%%blockSize) + 1):n
```

```
        base1 <- base[aux, ]
        bound1 <- bound[aux, ]
        if (add.model != "SBM") {
            eff <- round(dea.gem(base = base1, noutput, fixed,
                rts, bound = bound1, add.model)$eff, 7)
            index <- which(is.na(eff))
            if (length(index) > 0) {
                eff[index] <- round(dea.gem(base = base1, noutput,
                  fixed, rts, bound = bound1, add.model, whichDMUs = index)$eff,
                  7)
            }
            baseEfficient[[i + 1]] <- base1[which(eff == 0),
                ]
        }
        else {
            eff <- round(dea.sbm(base = base1, noutput, fixed,
                rts, bound = bound1)$eff, 7)
            index <- which(is.na(eff))
            if (length(index) > 0) {
                eff[index] <- round(dea.sbm(base = base1, noutput,
                  fixed, rts, bound = bound1, whichDMUs = index)$eff,
                  7)
            }
            baseEfficient[[i + 1]] <- base1[which(eff == 1),
                ]
        }
    }
    baseEfficient <- do.call("rbind", baseEfficient)
    if (add.model != "SBM") {
        eff <- round(dea.gem(base = base1, noutput, fixed, rts,
            bound = bound1, add.model)$eff, 7)
        index <- which(is.na(eff))
        if (length(index) > 0) {
            eff[index] <- round(dea.gem(base = base1, noutput,
                fixed, rts, bound = bound1, add.model, whichDMUs = index)$eff,
                7)
        }
        baseEfficient <- base1[which(eff == 0), ]
    }
    else {
        eff <- round(dea.sbm(base = base1, noutput, fixed, rts,
            bound = bound1)$eff, 7)
        index <- which(is.na(eff))
        if (length(index) > 0) {
            eff[index] <- round(dea.sbm(base = base1, noutput,
                fixed, rts, bound = bound1, whichDMUs = index)$eff,
                7)
        }
        baseEfficient <- base1[which(eff == 1), ]
    }
    eff <- list()
    for (i in 1:mod) {
        aux <- blocks[i]:(blocks[i + 1] - 1)
```

```
        base1 <- base[aux, ]
        base1 <- rbind(base1, baseEfficient)
        bound1 <- bound[aux, ]
        if (!is.null(bound)) {
            df <- data.frame(matrix(0, nrow = nrow(base1[1:(nrow(base1) -
                blockSize), ]), ncol = ncol(base1)))
            names(df) <- names(bound1)
            bound1 <- rbind(bound1, df)
        }
        if (add.model != "SBM") {
            eff[[i]] <- dea.gem(base = base1, noutput, fixed,
                rts, bound = bound1, add.model, whichDMUs = 1:blockSize)$eff
            index <- which(is.na(eff[[i]]))
            if (length(index) > 0) {
                eff[[i]][index] <- dea.gem(base = base1, noutput,
                  fixed, rts, bound = bound1, add.model, whichDMUs = index)$eff
            }
        }
        else {
            eff[[i]] <- dea.sbm(base = base1, noutput, fixed,
                rts, bound = bound1, whichDMUs = 1:blockSize)$eff
            index <- which(is.na(eff[[i]]))
            if (length(index) > 0) {
                eff[[i]][index] <- dea.sbm(base = base1, noutput,
                  fixed, rts, bound = bound1, whichDMUs = index)$eff
            }
        }
    }
    if (n%%blockSize != 0) {
        aux <- (n - (n%%blockSize) + 1):n
        base1 <- base[aux, ]
        base1 <- rbind(base1, baseEfficient)
        bound1 <- bound[aux, ]
        newBlockSize <- nrow(base) - mod * blockSize
        if (!is.null(bound)) {
            df <- data.frame(matrix(0, nrow = nrow(base1[1:(nrow(base1) -
                newBlockSize), ]), ncol = ncol(base1)))
            names(df) <- names(bound1)
            bound1 <- rbind(bound1, df)
        }
        if (add.model != "SBM") {
            eff[[i + 1]] <- dea.gem(base = base1, noutput, fixed,
                rts, bound = bound1, add.model, whichDMUs = 1:newBlockSize)$eff
            index <- which(is.na(eff[[i + 1]]))
            if (length(index) > 0) {
                eff[[i + 1]][index] <- dea.gem(base = base1,
                  noutput, fixed, rts, bound = bound1, add.model,
                  whichDMUs = index)$eff
            }
        }
        else {
            eff[[i + 1]] <- dea.sbm(base = base1, noutput, fixed,
                rts, bound = bound1, whichDMUs = 1:newBlockSize)$eff
```

```
            index <- which(is.na(eff[[i + 1]]))
            if (length(index) > 0) {
                eff[[i + 1]][index] <- dea.sbm(base = base1,
                  noutput, fixed, rts, bound = bound1, whichDMUs = index)$eff
            }
        }
    }
    eff <- unlist(eff)
    return(eff)
  }
```

---

dea.gem *Generalized Efficiency Measures (Additive DEA Models)*

Description

Calculate additive Data Envelopment Analysis (DEA) efficiency with five Generalized Efficiency Measures (GEM): unweighted additive model (Cooper et al., 2007), Range Adjusted Measure (Cooper et al., 1999; 2001), Bounded Adjusted Measure (Cooper et al., 2011), Measure of Inefficiency Proportions (Cooper et al., 1999), and the Lovell-Pastor Measure (Lovell and Pastor, 1995).

Usage

```
dea.gem(base, noutput, fixed = NULL, rts = 2, bound = NULL,
  add.model = c("additive", "RAM", "BAM", "MIP", "LovPast"),
  whichDMUs = NULL, print.status = FALSE)
```

Arguments

base	A data frame with N rows and S+M columns, where N is the number of Decision-Making Units (DMUs), S is the number of outputs and M is the number of inputs.
noutput	The number of outputs produced by the DMUs. All DMUs must produce the same number of outputs.
fixed	A numeric vector containing column indices for fixed (non-controllable) outputs and/or inputs (if any) in the data. Defaults to NULL.
rts	Returns to scale specification. 1 for constant returns to scale and 2 (default) for variable returns to scale.
bound	A data frame with N rows and S+M columns containing user-defined bounds on the slacks of each DMU. If bounds are supplied by the user in cases where some outputs and/or inputs are fixed, values should be 0 for these fixed variables. Same for slacks that do not require bounds. Defaults to NULL.
add.model	Additive model to calculate efficiency. additive: unweighted additive model (Cooper et al., 2007); RAM: Range Adjusted Measure (Cooper et al., 1999; 2001); BAM: Bounded Adjusted Measure (Cooper et al., 2011); MIP: Measure of Inefficiency Proportions (Cooper et al., 1999); LovPast: the Lovell-Pastor Measure (Lovell and Pastor, 1995).

whichDMUs	Numeric vector specifying the line numbers of the DMUs for which efficiency should be calculated. Defaults to NULL, i.e. by default efficiency is calculated for all DMUs in the dataset.
print.status	Defaults to FALSE. If the solution of the linear program is NA for one or more DMUs, print.status can be set to TRUE to find out why.

Details

Generalized Efficiency Measures (GEMs) are additive DEA models that maximize the sum of input and output slacks for each DMU. This sum is an aggregate measure of all inefficiencies that a DMU may exhibit in its inputs and outputs and can thus be seen as a holistic measure of (Pareto-Koopmans) inefficiency (efficient DMUs have no inefficiencies, thus their sum of slacks is 0). GEMs differ in that they weigh slacks in the objective function in different manners: the unweighted additive model does not in fact weigh the slacks, i.e. all slacks are assigned a trivial weight of 1 (Cooper et al., 2007); RAM weighs input and output slacks by the ranges in inputs and outputs respectively (Cooper et al., 1999). BAM weighs input and output slacks by the lower-sided ranges in inputs and the upper-sided ranges in outputs respectively (Cooper et al., 2011). MIP weighs input and output slacks by the respective inputs used and outputs produced by each DMU (Cooper et al., 1999); the Lovell-Pastor Measure weighs input and output slacks by the standard deviations in inputs and outputs respectively (Lovell and Pastor, 1995).

Models RAM, BAM, MIP and the Lovell-Pastor measure are units invariant, that is, the value of the aggregate inefficiency score of a DMU is independent of the units in which the inputs and outputs are measured, as long as these units are the same for every DMU. When a variable returns to scale specification is assumed, the unweighted additive model, RAM, BAM and the Lovell-Pastor measure are translation invariant, that is, they can accommodate zero or negative values of inputs and outputs.

Value

If print.status=FALSE, returns a data frame with N rows and 1+N+S+M columns. Column 1 reports the inefficiency score of each DMU. Columns 2 to N+1 contain the lambda values (intensity variables) indicating the DMU(s) that serve as reference for each DMU. Columns 1+N+1 to 1+N+S report the optimal output slacks for each DMU. Columns 1+N+S+1 to 1+N+S+M report the optimal input slacks for each DMU. When the linear program of a DMU is infeasible, the row of this DMU in the data frame will have NA values.

If print.status=TRUE, returns a list with two elements. The first element is the aforementioned data frame. The second element is a numeric vector indicating the solution status of the linear program (e.g. 0: solution found; 2: problem is infeasible; 5: problem needs scaling; etc. See https://CRAN.R-project.org/package=lpSolveAPI).

Note

Models RAM and BAM return an inefficiency score that is bounded by 0 and 1. Subtracting this inefficiency score from 1 gives an efficiency score that is also bounded by 0 and 1, with 1 indicating that the DMU under evaluation is fully efficient (zero slacks). The other GEMs do not carry this property, unless the slacks are bounded appropriately. For instance, the user may demand that output slacks are bounded by the actual outputs produced by each DMU, resulting in a MIP measure of inefficiency that, when divided by S+M and subtracted from 1, can be converted to an efficiency

measure that is bounded by 0 and 1. Note that with RAM and BAM the bounds must be smaller than, or equal to, the ranges (for RAM) or the sided ranges (for BAM) for the inefficiency scores to be bounded by 0 and 1.

Author(s)

Andreas Diomedes Soteriades, <andreassot10@yahoo.com>

References

Cooper W. W., Park K. S., 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

Cooper W. W., Park K. S., Pastor J. T. (2001) The range adjusted measure (RAM) in DEA: a response to the comment by Steinmann and Zweifel. *Journal of Productivity Analysis*, **15**, 145–152

Cooper W. W., Pastor J. T., Borras F., Aparicio J., Pastor D. (2011) BAM: a bounded adjusted measure of efficiency for use with bounded additive models. *Journal of Productivity Analysis*, **35**, 85–94

Cooper W. W., Seiford L., Tone K. (2007) *Data Envelopment Analysis: a comprehensive text with models, applications, references and DEA-Solver software*. New York: Springer

Lovell, C. A. K., Pastor J. T. (1995) Units invariant and translation invariant DEA models. *Operations Research Letters*, **18**, 147–151

See Also

dea.sbm, dea.fast

Examples

```
# Twelve DMUs, 2 inputs, 2 outputs
# (see Table 1.5 in Cooper et al., 2007):
base <- data.frame(
  y1= c(100,150,160,180,94,230,220,152,190,250,260,250),
  y2= c(90,50,55,72,66,90,88,80,100,100,147,120),
  x1= c(20,19,25,27,22,55,33,31,30,50,53,38),
  x2= c(151,131,160,168,158,255,235,206,244,268,306,284))

# Example 1: Get inefficiency scores,
# lambdas and slacks for all DMUs, with each GEM:
models <- c("additive", "RAM", "BAM", "MIP", "LovPast")
for(i in models) {
  print(i)
  results <- dea.gem(base, noutput= 2, add.model= i)
  print(results)
}

# Example 2: Same as above, but consider output y2 and input x1 as fixed:
for(i in models) {
  print(i)
  results <- dea.gem(base, noutput= 2, add.model= i, fixed= c(2, 3))
```

```
  print(results)
}

# Example 3: Impose upper bounds to slacks and get BAM inefficiency:
# upper-sided range for output:
Uo <- sweep(-base[c(1,2)], 2,
  apply(base[c(1,2)], 2, max), '+')
# lower-sided range for input
Li <- sweep(base[c(3,4)], 2,
  apply(base[c(3,4)], 2, min), '-')
# ensure bounds are <= the sided ranges:
bound <- cbind(Uo, Li)/2
# results with bounds:
dea.gem(base, noutput= 2, add.model= "BAM", bound= bound)$eff
# removing bounds allows for larger inefficiencies:
dea.gem(base, noutput= 2, add.model= "BAM", bound= NULL)$eff
# check solution status of linear programs when y2 and x1 are fixed
# and y1, x2 slacks are bounded:
fixed <- c(2,3)
bound[fixed] <- 0
for(i in models) {
  print(i)
  results <- dea.gem(base, noutput= 2, add.model= i,
    bound= bound, fixed= c(2, 3), rts= 1, print.status= TRUE)[[2]]
  print(results)
}

# Example 4: Get inefficiency scores for DMUs 11 and 12:
bound <- base
fixed <- c(2,3)
bound[fixed] <- 0
for(i in models) {
  print(i)
  results <- dea.gem(base, noutput= 2, add.model= i,
    bound= base, fixed= c(2, 3), rts= 1, whichDMUs= c(11, 12))$eff
  print(results)
}

# Example 5: a typical scaling problem in linear programing
# and how to deal with it:
# get data from package Benchmarking:
library(Benchmarking)
data(pigdata)
base <- pigdata[, 2:9][, c(7,8,1:6)]
results <- dea.gem(base, noutput= 2, rts= 1,
  add.model= 'RAM', print.status= TRUE)
# inefficiency for DMU 37 is NA:
which(is.na(results[[1]]$eff))
# error status: 5, i.e. scaling problem:
results[[2]][37]
# scale data:
results <- dea.gem(base/1000, noutput= 2, rts= 1,
  add.model= 'RAM', print.status= TRUE)
```

```
which(is.na(results[[1]]$eff)) # problem solved!

## The function is currently defined as
function (base, noutput, fixed = NULL, rts = 2, bound = NULL,
    add.model = c("additive", "RAM", "BAM", "MIP", "LovPast"),
    whichDMUs = NULL, print.status = FALSE)
{
    s <- noutput
    m <- ncol(base) - s
    n <- nrow(base)
    ifelse(!is.null(whichDMUs), nn <- length(whichDMUs), nn <- n)
    if (is.null(whichDMUs)) {
        whichDMUs <- 1:n
    }
    re <- data.frame(matrix(0, nrow = nn, ncol = 1 + n + s +
        m))
    names(re) <- c("eff", paste("lambda", 1:n, sep = ""), paste("slack.y",
        1:s, sep = ""), paste("slack.x", 1:m, sep = ""))
    lpmodel <- make.lp(nrow = 0, ncol = n + s + m)
    slacks <- diag(s + m)
    slacks[1:s, ] <- -slacks[1:s, ]
    type <- rep("=", s + m)
    if (!is.null(fixed)) {
        slacks[, fixed] <- 0
        type[fixed[fixed <= s]] <- ">="
        type[fixed[fixed > s]] <- "<="
    }
    A <- cbind(t(base), slacks)
    for (i in 1:(s + m)) {
        add.constraint(lpmodel, xt = A[i, ], type = type[i],
            rhs = 0)
    }
    if (add.model == "BAM") {
        Uo <- t(apply(data.frame(base[, 1:s]), 2, max) - t(base[,
            1:s]))
        Li <- t(t(base[, (s + 1):(s + m)]) - apply(data.frame(base[,
            (s + 1):(s + m)]), 2, min))
        if (!is.null(bound) & rts == 1) {
            index1 <- which(colSums(bound) == 0)
            bound[, index1] <- data.frame(Uo, Li)[, index1]
            if (!is.null(fixed)) {
                bound[fixed] <- 0
            }
        }
        if (is.null(bound) & rts == 1) {
            bound <- data.frame(Uo, Li)
            if (!is.null(fixed)) {
                bound[fixed] <- 0
            }
        }
    }
    if (add.model == "RAM") {
        Ro <- apply(data.frame(base[, 1:s]), 2, max) - apply(data.frame(base[,
```

```
        1:s]), 2, min)
    Ri <- apply(data.frame(base[, (s + 1):(s + m)]), 2, max) -
        apply(data.frame(base[, (s + 1):(s + m)]), 2, min)
    if (!is.null(bound) & rts == 1) {
        index1 <- which(colSums(bound) == 0)
        bound[, index1] <- t(as.data.frame(matrix(c(Ro, Ri),
            nrow = s + m, ncol = nrow(base))))[, index1]
        if (!is.null(fixed)) {
            bound[fixed] <- 0
        }
    }
    if (is.null(bound) & rts == 1) {
        bound <- t(as.data.frame(matrix(c(Ro, Ri), nrow = s +
            m, ncol = nrow(base))))
        if (!is.null(fixed)) {
            bound[fixed] <- 0
        }
    }
}
if (!is.null(bound)) {
    index <- which(colSums(bound) != 0)
    nrows <- length(index)
    A.bound <- matrix(0, nrow = nrows, ncol = n + s + m)
    k <- 0
    for (i in index) {
        k <- k + 1
        A.bound[k, n + index[k]] <- 1
    }
    A <- rbind(A, A.bound)
    for (i in 1:k) {
        add.constraint(lpmodel, xt = A[s + m + i, ], type = "<=",
            rhs = 0)
    }
}
syx <- rep(-1, s + m)
syx[fixed] <- 0
if (add.model == "additive") {
    set.objfn(lpmodel, c(rep(0, n), syx))
}
if (add.model == "RAM") {
    Ro[Ro == 0] <- Inf
    Ri[Ri == 0] <- Inf
    Ranges <- (1/c(Ro, Ri))/abs(sum(syx))
    set.objfn(lpmodel, c(rep(0, n), as.numeric(syx * Ranges)))
}
if (add.model == "LovPast") {
    st.dev <- apply(base, 2, sd)
    st.dev[st.dev == 0] <- Inf
    set.objfn(lpmodel, c(rep(0, n), syx/st.dev))
}
k <- 0
if (print.status == TRUE) {
    ifelse(!is.null(whichDMUs), solution.status <- rep(0,
```

```
        nn), solution.status <- rep(0, n))
    }
    if (rts == 2 & is.null(bound)) {
        add.constraint(lpmodel, xt = c(rep(1, n), rep(0, s +
            m)), type = "=", rhs = 0)
        for (i in whichDMUs) {
            k <- k + 1
            if (add.model == "MIP") {
                set.objfn(lpmodel, c(rep(0, n), as.numeric(syx/base[i,
                  ])))
            }
            if (add.model == "BAM") {
                Uo[i, ][Uo[i, ] == 0] <- Inf
                Li[i, ][Li[i, ] == 0] <- Inf
                Ranges <- (1/c(Uo[i, ], Li[i, ]))/abs(sum(syx))
                set.objfn(lpmodel, c(rep(0, n), as.numeric(syx *
                  Ranges)))
            }
            set.rhs(lpmodel, b = as.numeric(c(base[i, ], 1)))
            x <- solve(lpmodel)
            if (print.status == TRUE) {
                solution.status[k] <- x
            }
            re[k, ] <- c(-get.objective(lpmodel), tail(get.primal.solution(lpmodel),
                s + m + n))
            if (x != 0) {
                re[k, ] <- rep(NA, ncol(re))
            }
        }
    }
    if (rts == 2 & !is.null(bound)) {
        add.constraint(lpmodel, xt = c(rep(1, n), rep(0, s +
            m)), type = "=", rhs = 0)
        for (i in whichDMUs) {
            k <- k + 1
            if (add.model == "MIP") {
                set.objfn(lpmodel, c(rep(0, n), as.numeric(syx/base[i,
                  ])))
            }
            if (add.model == "BAM") {
                Uo[i, ][Uo[i, ] == 0] <- Inf
                Li[i, ][Li[i, ] == 0] <- Inf
                Ranges <- (1/c(Uo[i, ], Li[i, ]))/abs(sum(syx))
                set.objfn(lpmodel, c(rep(0, n), as.numeric(syx *
                  Ranges)))
            }
            if (add.model %in% c("RAM", "BAM")) {
                set.rhs(lpmodel, b = as.numeric(c(base[i, ],
                  bound[i, index], 1)))
            }
            if (sum(add.model == c("RAM", "BAM")) == 0) {
                bound[bound == 0] <- 10^10
                set.rhs(lpmodel, b = as.numeric(c(base[i, ],
```

```
                bound[i, index], 1)))
            }
            solve(lpmodel)
            x <- solve(lpmodel)
            if (print.status == TRUE) {
                solution.status[k] <- x
            }
            re[k, ] <- c(-get.objective(lpmodel), tail(get.primal.solution(lpmodel),
                s + m + n))
            if (x != 0) {
                re[k, ] <- rep(NA, ncol(re))
            }
        }
    }
    if (rts == 1 & is.null(bound)) {
        for (i in whichDMUs) {
            k <- k + 1
            if (add.model == "MIP") {
                set.objfn(lpmodel, c(rep(0, n), as.numeric(syx/base[i,
                  ])))
            }
            set.rhs(lpmodel, b = as.numeric(base[i, ]))
            x <- solve(lpmodel)
            if (print.status == TRUE) {
                solution.status[k] <- x
            }
            re[k, ] <- c(-get.objective(lpmodel), tail(get.primal.solution(lpmodel),
                s + m + n))
            if (x != 0) {
                re[k, ] <- rep(NA, ncol(re))
            }
        }
    }
    if (rts == 1 & !is.null(bound)) {
        for (i in whichDMUs) {
            k <- k + 1
            if (add.model == "MIP") {
                set.objfn(lpmodel, c(rep(0, n), as.numeric(syx/base[i,
                  ])))
            }
            if (add.model == "BAM") {
                Uo[i, ][Uo[i, ] == 0] <- Inf
                Li[i, ][Li[i, ] == 0] <- Inf
                Ranges <- (1/c(Uo[i, ], Li[i, ]))/abs(sum(syx))
                set.objfn(lpmodel, c(rep(0, n), as.numeric(syx *
                  Ranges)))
            }
            if (add.model %in% c("RAM", "BAM")) {
                set.rhs(lpmodel, b = as.numeric(c(base[i, ],
                  bound[i, index])))
            }
            if (sum(add.model == c("RAM", "BAM")) == 0) {
                bound[bound == 0] <- 10^10
```

```
                set.rhs(lpmodel, b = as.numeric(c(base[i, ],
                  bound[i, index])))
            }
            x <- solve(lpmodel)
            if (print.status == TRUE) {
                solution.status[k] <- x
            }
            ifelse(x != 0, re[k, ] <- rep(NA, ncol(re)), re[k,
                ] <- c(-get.objective(lpmodel), tail(get.primal.solution(lpmodel),
                s + m + n)))
        }
    }
    if (!is.null(fixed)) {
        re <- re[, -(1 + n + fixed)]
    }
    if (print.status == TRUE) {
        reList <- list()
        reList[[1]] <- re
        reList[[2]] <- solution.status
        names(reList[[2]]) <- whichDMUs
        re <- reList
    }
    return(re)
}
```

---

dea.sbm *Slacks-Based Measure (SBM) of Efficiency*

Description

Calculate additive Data Envelopment Analysis (DEA) efficiency with the Slacks-Based Measure (SBM) of efficiency (Tone, 2001)

Usage

```
dea.sbm(base, noutput, fixed = NULL, rts = 2,
  bound = NULL, whichDMUs = NULL, print.status = FALSE)
```

Arguments

base	A data frame with N rows and S+M columns, where N is the number of Decision-Making Units (DMUs), S is the number of outputs and M is the number of inputs.
noutput	The number of outputs produced by the DMUs. All DMUs must produce the same number of outputs.
fixed	A numeric vector containing column indices for fixed (non-controllable) outputs and/or inputs (if any) in the data. Defaults to NULL.
rts	Returns to scale specification. 1 for constant returns to scale and 2 (default) for variable returns to scale.

bound	A data frame with N rows and S+M columns containing user-defined bounds on the slacks of each DMU. If bounds are supplied by the user in cases where some outputs and/or inputs are fixed, values should be 0 for these fixed variables. Same for slacks that do not require bounds. Defaults to NULL.
whichDMUs	Numeric vector specifying the line numbers of the DMUs for which efficiency should be calculated. Defaults to NULL, i.e. by default efficiency is calculated for all DMUs in the dataset.
print.status	Defaults to FALSE. If the solution of the linear program is NA for one or more DMUs, print.status can be set to TRUE to find out why.

Details

The Slacks-Based Measure (SBM) of efficiency (Tone, 2001) is an additive DEA model that maximizes the sum of input and output slacks for each DMU. Unlike other additive DEA models, SBM's objective function has a ratio form, with input slacks summed in the numerator and output slacks summed in the denominator. These sums in the ratio result in an aggregate measure of all inefficiencies that a DMU may exhibit in its inputs and outputs and can thus be seen as a holistic measure of (Pareto-Koopmans) efficiency. SBM weighs input and output slacks in the objective function with the respective inputs used and outputs produced by each DMU. SBM is units invariant, that is, the value of the aggregate inefficiency score of a DMU is independent of the units in which the inputs and outputs are measured, as long as these units are the same for every DMU. For each DMU, SBM returns an efficiency score that is bounded by 0 and 1.

Value

If print.status=FALSE, returns a data frame with N rows and 1+N+S+M columns. Column 1 reports the inefficiency score of each DMU. Columns 2 to N+1 contain the lambda values (intensity variables) indicating the DMU(s) that serve as reference for each DMU. Columns 1+N+1 to 1+N+S report the optimal output slacks for each DMU. Columns 1+N+S+1 to 1+N+S+M report the optimal input slacks for each DMU. When the linear program of a DMU is infeasible, the row of this DMU in the data frame will have NA values.

If print.status=TRUE, returns a list with two elements. The first element is the aforementioned data frame. The second element is a numeric vector indicating the solution status of the linear program (e.g. 0: solution found; 2: problem is infeasible; 5: problem needs scaling; etc. See https://CRAN.R-project.org/package=lpSolveAPI).

Author(s)

Andreas Diomedes Soteriades, <andreassot10@yahoo.com>

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.gem, dea.fast

Examples

```
# Twelve DMUs, 2 inputs, 2 outputs
# (see Table 1.5 in Cooper et al., 2007):
base <- data.frame(
  y1= c(100,150,160,180,94,230,220,152,190,250,260,250),
  y2= c(90,50,55,72,66,90,88,80,100,100,147,120),
  x1= c(20,19,25,27,22,55,33,31,30,50,53,38),
  x2= c(151,131,160,168,158,255,235,206,244,268,306,284))

# Example 1: Get inefficiency scores,
# lambdas and slacks for all DMUs:
dea.sbm(base, noutput= 2, rts= 1)
dea.sbm(base, noutput= 2, rts= 2)

# Example 2: Same as above, but consider output y2 and input x1 as fixed:
dea.sbm(base, noutput= 2, rts= 1, fixed= c(2,3))
dea.sbm(base, noutput= 2, rts= 2, fixed= c(2,3))

# Example 3: Impose an upper bound to all slacks:
# results with bounds
dea.sbm(base, noutput= 2, rts= 1, bound= base/12)$eff
# removing bounds allows for larger inefficiencies:
dea.sbm(base, noutput= 2, rts= 1, bound= NULL)$eff
# check solution status of linear programs when y2 and x1 are fixed
# and y1, x2 slacks are bounded:
bound <- base
fixed <- c(2,3)
bound[fixed] <- 0
dea.sbm(base, noutput= 2, bound= bound,
  fixed= c(2, 3), rts= 1, print.status= TRUE)[[2]]
dea.sbm(base, noutput= 2, bound= bound,
  fixed= c(2, 3), rts= 2, print.status= TRUE)[[2]]

# Example 4: Get inefficiency scores for DMUs 11 and 12:
bound <- base
fixed <- c(2,3)
bound[fixed] <- 0
dea.sbm(base, noutput= 2, bound= bound,
  fixed= c(2, 3), rts= 1, whichDMUs= c(11, 12))$eff
dea.sbm(base, noutput= 2, bound= bound,
  fixed= c(2, 3), rts= 2, whichDMUs= c(11, 12))$eff

## The function is currently defined as
function (base, noutput, fixed = NULL, rts = 2, bound = NULL,
    whichDMUs = NULL, print.status = FALSE)
{
    s <- noutput
    m <- ncol(base) - s
    n <- nrow(base)
    ifelse(!is.null(whichDMUs), nn <- length(whichDMUs), nn <- n)
    if (is.null(whichDMUs)) {
        whichDMUs <- 1:n
```

```
}
re <- data.frame(matrix(0, nrow = nn, ncol = 1 + n + s +
    m))
names(re) <- c("eff", paste("lambda", 1:n, sep = ""), paste("slack.y",
    1:s, sep = ""), paste("slack.x", 1:m, sep = ""))
slacks <- diag(s + m)
slacks[1:s, ] <- -slacks[1:s, ]
type <- rep("=", s + m)
if (!is.null(fixed)) {
    slacks[, fixed] <- 0
    type[fixed[fixed <= s]] <- ">="
    type[fixed[fixed > s]] <- "<="
}
k <- 0
A.aux <- cbind(t(base), slacks)
index.fixed.y <- fixed[which(fixed %in% 1:s)]
index.fixed.x <- fixed[which(fixed %in% (s + 1):(s + m))]
S <- s - length(index.fixed.y)
M <- m - length(index.fixed.x)
if (print.status == TRUE) {
    ifelse(!is.null(whichDMUs), solution.status <- rep(0,
        nn), solution.status <- rep(0, n))
}
if (rts == 2 & is.null(bound)) {
    for (i in whichDMUs) {
        k <- k + 1
        lpmodel <- make.lp(nrow = 0, ncol = 1 + n + s + m)
        A <- cbind(-t(base)[, i], A.aux)
        xt <- as.numeric((1/S) * (1/base[i, 1:s]))
        if (!is.null(fixed)) {
            xt[index.fixed.y] <- 0
        }
        xt <- c(1, rep(0, n), xt, rep(0, m))
        add.constraint(lpmodel, xt = xt, type = "=", rhs = 0)
        for (j in 1:(s + m)) {
            add.constraint(lpmodel, xt = A[j, ], type = type[j],
              rhs = 0)
        }
        add.constraint(lpmodel, xt = c(-1, rep(1, n), rep(0,
            s + m)), type = "=", rhs = 0)
        set.rhs(lpmodel, b = c(1, rep(0, s + m + 1)))
        obj <- as.numeric((-1/M) * (1/base[i, (s + 1):(s +
            m)]))
        if (!is.null(fixed)) {
            obj[index.fixed.x - s] <- 0
        }
        obj <- c(1, rep(0, n + s), obj)
        set.objfn(lpmodel, obj = obj)
        x <- solve(lpmodel)
        if (print.status == TRUE) {
            solution.status[k] <- x
        }
        re[k, ] <- c(get.objective(lpmodel), tail(get.primal.solution(lpmodel),
```

```
            n + s + m)/get.primal.solution(lpmodel)[1 + 1 +
            s + m + 1 + 1])
        if (x != 0) {
            re[k, ] <- rep(NA, ncol(re))
        }
    }
}
if (rts == 2 & !is.null(bound)) {
    index <- which(colSums(bound) != 0)
    nrows <- length(index)
    A.bound <- matrix(0, nrow = nrows, ncol = n + s + m)
    kk <- 0
    for (i in index) {
        kk <- kk + 1
        A.bound[kk, n + index[kk]] <- 1
    }
    A.bound <- cbind(0, A.bound)
    for (i in whichDMUs) {
        A.bound <- matrix(0, nrow = nrows, ncol = n + s +
            m)
        kk <- 0
        for (j in index) {
            kk <- kk + 1
            A.bound[kk, n + index[kk]] <- 1
        }
        A.bound <- cbind(0, A.bound)
        k <- k + 1
        lpmodel <- make.lp(nrow = 0, ncol = 1 + n + s + m)
        A <- cbind(-t(base)[, i], A.aux)
        A.bound[, 1] <- as.numeric(-bound[i, index])
        A.bound[A.bound[, 1] == 0, ] <- 0
        A <- rbind(A, A.bound)
        xt <- as.numeric((1/S) * (1/base[i, 1:s]))
        if (!is.null(fixed)) {
            xt[index.fixed.y] <- 0
        }
        xt <- c(1, rep(0, n), xt, rep(0, m))
        add.constraint(lpmodel, xt = xt, type = "=", rhs = 0)
        for (j in 1:(s + m)) {
            add.constraint(lpmodel, xt = A[j, ], type = type[j],
              rhs = 0)
        }
        add.constraint(lpmodel, xt = c(-1, rep(1, n), rep(0,
            s + m)), type = "=", rhs = 0)
        for (l in 1:kk) {
            add.constraint(lpmodel, xt = A[s + m + l, ],
              type = "<=", rhs = 0)
        }
        set.rhs(lpmodel, b = c(1, rep(0, s + m + 1 + l)))
        obj <- as.numeric((-1/M) * (1/base[i, (s + 1):(s +
            m)]))
        if (!is.null(fixed)) {
            obj[index.fixed.x - s] <- 0
```

```
        }
        obj <- c(1, rep(0, n + s), obj)
        set.objfn(lpmodel, obj = obj)
        x <- solve(lpmodel)
        if (print.status == TRUE) {
            solution.status[k] <- x
        }
        re[k, ] <- c(get.objective(lpmodel), tail(get.primal.solution(lpmodel),
            n + s + m)/get.primal.solution(lpmodel)[1 + 1 +
            s + m + 1 + sum(colSums(bound) != 0) + 1])
        if (x != 0) {
            re[k, ] <- rep(NA, ncol(re))
        }
    }
}
if (rts == 1 & is.null(bound)) {
    for (i in whichDMUs) {
        k <- k + 1
        lpmodel <- make.lp(nrow = 0, ncol = 1 + n + s + m)
        A <- cbind(-t(base)[, i], A.aux)
        xt <- as.numeric((1/S) * (1/base[i, 1:s]))
        if (!is.null(fixed)) {
            xt[index.fixed.y] <- 0
        }
        xt <- c(1, rep(0, n), xt, rep(0, m))
        add.constraint(lpmodel, xt = xt, type = "=", rhs = 0)
        for (j in 1:(s + m)) {
            add.constraint(lpmodel, xt = A[j, ], type = type[j],
              rhs = 0)
        }
        set.rhs(lpmodel, b = c(1, rep(0, s + m)))
        obj <- as.numeric((-1/M) * (1/base[i, (s + 1):(s +
            m)]))
        if (!is.null(fixed)) {
            obj[index.fixed.x - s] <- 0
        }
        obj <- c(1, rep(0, n + s), obj)
        set.objfn(lpmodel, obj = obj)
        x <- solve(lpmodel)
        if (print.status == TRUE) {
            solution.status[k] <- x
        }
        re[k, ] <- c(get.objective(lpmodel), tail(get.primal.solution(lpmodel),
            n + s + m)/get.primal.solution(lpmodel)[1 + 1 +
            s + m + 1])
        if (x != 0) {
            re[k, ] <- rep(NA, ncol(re))
        }
    }
}
if (rts == 1 & !is.null(bound)) {
    index <- which(colSums(bound) != 0)
    nrows <- length(index)
```

```
        A.bound <- matrix(0, nrow = nrows, ncol = n + s + m)
        kk <- 0
        for (i in index) {
            kk <- kk + 1
            A.bound[kk, n + index[kk]] <- 1
        }
        A.bound <- cbind(0, A.bound)
        for (i in whichDMUs) {
            k <- k + 1
            lpmodel <- make.lp(nrow = 0, ncol = 1 + n + s + m)
            A <- cbind(-t(base)[, i], A.aux)
            A.bound[, 1] <- as.numeric(-bound[i, index])
            A.bound[A.bound[, 1] == 0, ] <- 0
            A <- rbind(A, A.bound)
            xt <- as.numeric((1/S) * (1/base[i, 1:s]))
            if (!is.null(fixed)) {
                xt[index.fixed.y] <- 0
            }
            xt <- c(1, rep(0, n), xt, rep(0, m))
            add.constraint(lpmodel, xt = xt, type = "=", rhs = 0)
            for (j in 1:(s + m)) {
                add.constraint(lpmodel, xt = A[j, ], type = type[j],
                  rhs = 0)
            }
            for (l in 1:kk) {
                add.constraint(lpmodel, xt = A[s + m + l, ],
                  type = "<=", rhs = 0)
            }
            set.rhs(lpmodel, b = c(1, rep(0, s + m + l)))
            obj <- as.numeric((-1/M) * (1/base[i, (s + 1):(s +
                m)]))
            if (!is.null(fixed)) {
                obj[index.fixed.x - s] <- 0
            }
            obj <- c(1, rep(0, n + s), obj)
            set.objfn(lpmodel, obj = obj)
            x <- solve(lpmodel)
            if (print.status == TRUE) {
                solution.status[k] <- x
            }
            re[k, ] <- c(get.objective(lpmodel), tail(get.primal.solution(lpmodel),
                n + s + m)/get.primal.solution(lpmodel)[1 + 1 +
                s + m + sum(colSums(bound) != 0) + 1])
            if (x != 0) {
                re[k, ] <- rep(NA, ncol(re))
            }
        }
    }
    if (!is.null(fixed)) {
        re <- re[, -(1 + n + fixed)]
    }
    if (print.status == TRUE) {
        reList <- list()
```

```
        reList[[1]] <- re
        reList[[2]] <- solution.status
        names(reList[[2]]) <- whichDMUs
        re <- reList
    }
    return(re)
}
```

Index