Package ‘DEAboot’

July 8, 2015

Type Package
Title FORTRAN input/output DEA bootstrapping
Version 1.11
Date 2015-07-08
Author Joe Atwood and Saleem Shaik
Maintainer Joe Atwood <jatwood@montana.edu>
Depends Benchmarking
Description Provides access to FORTRAN code for input/output DEA bootstrapping.
License GPL-2
LazyLoad yes
Archs i386, x64

R topics documented:

DEAboot-package .......................................................... 1
dea.boot_DEAboot ......................................................... 3
dea.sample_DEAboot ....................................................... 4
DEAboot ................................................................. 5
MMLP ................................................................. 8
normgen ............................................................... 13
sampleum ............................................................. 14
seconds .............................................................. 15
ugen .............................................................. 15

Index

DEAboot-package DEAboot

Description

Uses FORTRAN to bootstrap input/output DEA models.
Also provides an interface to Morris and Miller’s open source linear programming Fortran code.

Details

1
Package: DEAboot
Type: Package
Version: 1.11
Date: 2015-07-08
License: GPL-2
LazyLoad: yes

Author(s)

R-Fortran bootstrapping and random number generation interface/fortran code: Joe Atwood

Alan Miller fortran code obtained from: http://jblevins.org/mirror/amiller/

Site Statement: "This is an archived copy of the Fortran source code repository of Alan Miller previously located at http://users.bigpond.net.au/amiller/. It is hosted by Jason Blevins with permission. The site has been slightly reformatted, but the source code and descriptions below have not been modified. All code written by Alan Miller is released into the public domain."

Fortran linear programming code listing:

"smplx.f90 Linear programming using the simplex algorithm. This is a translation of the Fortran 66 program from the NSWC (Naval Surface Warfare Center) library written by Alfred Morris. There is also a simple test program t_smplx.f90. Needs the module constant.f90 which defines the precision and returns certain machine constants."

Fortran random number generation code listing:

"mt19937.f90 The 'Mersenne Twister' random number generator from Japan with a cycle of length \((2^{19937} - 1)\). mt19937a.f90 is a version for compilers which stop when there are integer overflows, as some do when compiler check options are enabled for debugging purposes. .... . mt19937.f90 was revised on 5 February 2002;"

GPL license statement contained in mt19937.f90 code

"This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details."
**Description**

The function `dea.boot_DEAboot` is borrowed and modified from the Benchmarking package to give the same answers as those obtained from the fortran code. We thank the authors of the Benchmarking package for allowing us to borrow and edit their code.

This function in included in the DEAboot package primarily to demonstrate that the Fortran code based call DEAboot generates results equivalent to those obtained from the Benchmarking package’s function `dea.boot` if `dea.boot` is modified to use the same random numbers.

The function DEAboot can be run independently from the function `dea.boot_DEAboot`.

**Usage**

```r
dea.boot_DEAboot(X, Y, NREP = 200, EFF = NULL, RTS = "vrs", ORIENTATION="in",
alpha = 0.05, XREF = NULL, YREF = NULL, EREF = NULL,
DIRECT = NULL, TRANSPOSE = FALSE, LP,printum=FALSE,printmod=5,saveum=FALSE)
```

**Arguments**

- **X**: Inputs of firms to be evaluated, a K x m matrix of observations of K firms with m inputs (firm x input)
- **Y**: Outputs of firms to be evaluated, a K x n matrix of observations of K firms with n outputs (firm x input).
- **NREP**: Number of bootstrap replicats
- **EFF**: Efficiencies for (X,Y) relative to the technology generated from (XREF,YREF).
- **RTS**: The returns to scale assumptions as in `dea`, only works for "vrs", "drs", and "crs"; more to come.
- **ORIENTATION**: Input efficiency "in" (1), output efficiency "out" (2), and graph efficiency "graph" (3).
- **alpha**: One minus the size of the confidence interval for the bias corrected efficiencies
- **XREF**: Inputs of the firms determining the technology, defaults to X.
- **YREF**: Outputs of the firms determining the technology, defaults to Y.
- **EREF**: Efficiencies for the firms in XREF, YREF.
- **DIRECT**: Does not yet work and is therefore not used.
- **TRANSPOSE**: Input and output matrices are K x m and K x n for the default value TRANSPOSE=FALSE; this is standard in R for statistical models. When TRANSPOSE=TRUE data matrices are m x K and n x K.
- **LP**: Only for debugging purposes.
- **printum**: printum==TRUE prints runtime progress reports
- **printmod**: if printum==TRUE, progress reports are printed every printmod’th iteration
- **saveum**: if saveum!=FALSE, returns extra data at end of function call
Details

See the Benchmarking package’s documentation of the dea.boot function for a complete description of the arguments returned by the dea.boot function.

Value

The returned values from both functions are as follows:

- **eff**: Efficiencies
- **eff.bc**: Bias-corrected efficiencies
- **bias**: An array of bootstrap bias estimates for the firms in X,Y
- **conf.int**: K x 2 matrix with confidence interval for the estimated efficiencies
- **var**: An array of bootstrap variance estimates for the firms in X,Y
- **boot**: The replica bootstrap estimates of the Farrell efficiencies, a K times NREP matrix. Note the bootstrap estimates are sorted for each firm.

Author(s)

The good stuff: Peter Bogetoft and Lars Otto <larsot23@gmail.com>
The bad stuff: Joe Atwood <jatwood@montana.edu>

References

Peter Bogetoft and Lars Otto; *Benchmarking with DEA, SFA, and R*; Springer 2011.

Cinzia Dario and L. Simar; *Advanced Robust and Nonparametric Methods in Efficiency Analysis*. Methodology and Applications; Springer 2007.


Examples

```r
#not run
```

### dea.sample_DEAboot

**Resample efficiency scores in DEAboot**

**Description**

The function `dea.sample_DEAboot` is borrowed and modified from the Benchmarking package to give the same answers as those obtained from the fortran code. We thank the authors of the Benchmarking package for allowing us to borrow and edit their code.

**Usage**

```r
dea.sample_DEAboot(e,h,K=NULL,seedval)
```
**Arguments**

- `e`  
  original DEA eff estimates
- `h`  
  kernal value
- `K`  
  Number of bootstrap replicats
- `seedval`  
  seedvalue for fortran random number generator

**Details**

See the Benchmarking package’s documentation of the dea.sample function for a complete description of the arguments returned by the dea.sample function.

**Value**

- `estar`  
  Resampled Efficiencies
- `seedval`  
  incremented seed value for fortran random number generator

**Author(s)**

The good stuff: Peter Bogetoft and Lars Otto <larsot23@gmail.com>

The bad stuff: Joe Atwood <jatwood@montana.edu>

**Examples**

```r
#not run
```

---

**DEAbout**

*Fortran bootstrapping of input/output DEA models*

**Description**

Fortran bootstrapping of input/output DEA models

**Usage**

```r
DEAbout(X,Y,orient='in',RTS='crs',nboot=250)
```

**Arguments**

- `X`  
  An nDMU x nX matrix of Input observations
- `Y`  
  An nDMU x nY matrix of Output observations
- `orient`  
  Input efficiency "in" output efficiency "out"
- `RTS`  
  Returns to Scale: "vrs","drs","crs", and "irs"
- `nboot`  
  Number of bootstraps to complete.
  
  nboot=0 calculates DEA efficiency scores for each dmu but does not bootstrap the results
**DEAboot**

**Value**

- **h** bootstrapping h value
- **effvals** Vector of Efficiency Scores
- **effstatus** Status of Efficiency Scores
  - indstat = 0 the problem was solved; indstat = 1 the problem has no solution; indstat = 2 itermax iterations were performed-more needed; indstat = 3 sufficient accuracy could not be maintained to solve the problem; indstat = 4 the problem has an unbounded solution; indstat = 5 input error detected; indstat = 6 the solution may have been obtained;
- **boot** nDMU by nboot matrix of bootstrapped efficiency scores
- **bootstatus** status matrix of bootstrapped efficiency scores

**Author(s)**

Joe Atwood and Saleem Shaik

**Examples**

```r
# not run
# #####################################################
# require(DEAboot)
# #####################################################
# set.seed(2015)
# graphics.off()
# #####################################################
# orient="in"    # 1=in, 2=out
# RTS="crs"      # 1=vrs, 2=drs, 3=crs, 4=irs
# #####################################################
# nDMU=250
# nboot=250
# # Define Cobb-Douglas technology with CRS
# b1=0.5
# b2=1-b1
# # Generate "input levels"
# x1=runif(nDMU,5,10)
# x2=runif(nDMU,5,10)
# # Generate "frontier" output levels
# y1=x1*b1*x2*(1-b1)
# # Generate "inverse output efficiency scores"
# eff0=seq(0.25,1.0,length.out=nDMU)
# # Contract output levels away from the efficient frontier
# y1=eff0*y1
# # Put input and output quantities into matrices X and Y
# X=as.matrix(cbind(x1,x2))
# Y=as.matrix(y1)
# # Call the Fortran based DEAboot function
# time1=seconds() #Note: This function is in the DEAboot package.
# tmp1=DEAboot(X,Y,orient=orient,RTS=RTS,nboot=nboot)
# time2=seconds()
```
DEAbboot

```
# Run modified benchmarking package bootstrapping code
# time 3 = seconds()
# tmp2 = dea.boot(DEAbboot(X, Y, NREP = nboot, RTS = RTS, ORIENTATION = orient))
# time 4 = seconds()
#
# Run modified benchmarking package bootstrapping code with internal status printing
# time 5 = seconds()
# tmp3 = dea.boot(DEAbboot(X, Y, NREP = nboot, RTS = RTS, ORIENTATION = orient, printnum = TRUE, printmod = 25))
# time 6 = seconds()

# Run benchmarking package bootstrapping code
# time 7 = seconds()
# tmp4 = dea.boot(X, Y, NREP = nboot, RTS = RTS, ORIENTATION = orient)
# time 8 = seconds()

# Contrast DEAbboot results to benchmarking results
# (h1 = tmp1$h)
# effhat1 = tmp1$effvals
# boot1 = tmp1$boot
# (h2 = tmp2$h)
# effhat2 = as.vector(tmp2$eff)
# boot2 = tmp2$boot

# summary(effhat1 - effhat2)
# summary(as.vector(boot1) - as.vector(boot2))
# (x11)
# plot(boot1, boot2)
# abline(h = TRUE, col = 2)
```

### DEAbboot computation time

```
# time 2 - time 1
# time 6 = 0.89
```

### Benchmarking computation times

```
# summary(as.vector(boot1) - as.vector(boot2))
# abline(h = TRUE, col = 2)
```

**MMLP**

Morris-Miller Fortran LP code interface

### Description
Morris-Miller Fortran LP code interface

### Usage
```r
MMLP(objtype='max', obj, A, rest, rhs, itermax=1000, nsims=1)
```

### Arguments
- **objtype**: character string ‘max’ or ‘min’
- **obj**: vector of objective coefficients
- **A**: matrix of constraint coefficients
- **rest**: character vector of constraint signs ‘<=’, ‘>=’, or ‘=’
- **rhs**: vector of RHS values
- **itermax**: maximal number of LP iterations
- **nsims**: number of repetitions before returning results

### Value
- **objval**: objective value
- **xvals**: If indstat = 0 or 6, xvals returns the solution, the slack, and the surplus variable levels
- **indstat**: indstat = 0 the problem was solved; indstat = 1 the problem has no solution; indstat = 2 itermax iterations were performed—more needed; indstat = 3 sufficient accuracy could not be maintained to solve the problem; indstat = 4 the problem has an unbounded solution; indstat = 5 input error detected; indstat = 6 the solution may have been obtained;

### Author(s)
R and Fortran interface, duals calculations: Joe Atwood
MMLP code:
WRITTEN BY ALFRED H. MORRIS JR.
NAVAL SURFACE WEAPONS CENTER
DAHLGREN, VIRGINIA
Examples

# not run
# ###################################################################################################################
# Example 1
# ###################################################################################################################
# Determine computation times required to solve the Wyndor example 100,000
# # times using loops with IpSolveAPI and MMLP and constrasting the
# # results to solving the Wyndor problem 100,000 times within the MMLP fortran code
# ###################################################################################################################
# rm(list=ls())
# ###################################################################################################################
# require(DEAboot)
# require(IpSolve)
# require(IpSolveAPI)
# ###################################################################################################################
# nsims=100000
# ###################################################################################################################
# #construct problem
# nr=3
# nc=2
# objtype='max'
# objmax=1
# obj=c(3,5)
# A=matrix(c(
#   c(1,0),
#   c(0,2),
#   c(3,2)
# ),3,2,byrow=T)
# b=c(4,12,18)
# rest=c(c('<','<=','<')
# ###################################################################################################################
#
# set up lpSolveAPI object
# LP_API=make.lp(nrow=nr,ncol=nc)
# lp.control(LP_API,sense=objtype)
# set.objfn(LP_API,obj)
# for(i in 1:nr){
#  set.row(LP_API,i,A[i,])
# }
# set.constr.type(LP_API,rest)
# set.rhs(LP_API,b)
# solve with lpSolve
# tmp=lp(objtype,obj,A,rest,b,compute.sens=TRUE)
# tmp$objval;tmp$solution;tmp$duals[1:nr]
# [1] 36
# [2] 6
# [1] 0.0 1.5 1.0
#
# solve with lpSolveAPI
# (status=solve(LP_API))
# [1] 0
# get.objective(LP_API);get.variables(LP_API);get.dual.solution(LP_API)[2:(nr+1)]
# [1] 36
# [2] 6
# [1] 0.0 1.5 1.0
#
# solve with MMLP
# tmp2=MMLP(objtype=obj,A=A,rest=rest,rhs=b)
# tmp2$objval;tmp2$xvals[1:nc];tmp2$duals
# [1] 36
# [2] 6
# [1] 0.0 1.5 1.0
#
#
# time to obtain nsims solutions
# time=seconds()
# lpSolveAPI with loops
# obj=0
# for(j in 1:nsims){
#  set.objfn(LP_API,Obj)
#  for(i in 1:nr){
#   set.row(LP_API,i,A[i,])
#  }
#  set.constr.type(LP_API,rest)
#  set.rhs(LP_API,b)
#  (status=solve(LP_API))
#  obj[j]=get.objective(LP_API)
# } end loop
MMLP

```r
# remove lpSolveAPI object
# delete.lp(LP_API)

# MMLP with loops
# obj2=0
# for(j2 in 1:nsims){
# tmp=MMLP(objtype=objtype,obj=obj,A=A,rest=rest,rhs=b)
# obj2[j2]=tmp$objval
# }

# MMLP with internal loops
# tmp=MMLP(objtype=objtype,obj=obj,A=A,rest=rest,rhs=b,nsims=nsims)

# Example 2: A more realistic example
# A DEA example that computes the efficiency score for each of 10000 DMUs. The example uses traditional looping
# with both lpSolve and MMLP and constrasts the "R" loop
# to the results of using fortran loops within DEAboot
# (without running the bootstraps i.e. by setting nloop = 0)
# to obtain the efficiency score estimates for each DMU.
# require(DEAboot)
# require(lpSolveAPI)
# set.seed(2015)
# nDMU=10000
# Define Cobb-Douglas technology with CRS
# b1=0.5
```
# b2=1-b1
#generate "input levels"
x1=runif(nDMU,5,10)
x2=runif(nDMU,5,10)
#generate "Frontier" output levels
y1=x1*b1+x2*(1-b1)
#generate "inverse output efficiency scores"
inv_eff0=seq(0.25,1,0.01,length.out=nDMU)
#Contract output levels away from the efficient frontier
y1=inv_eff0*y1
eff0=1/inv_eff0
#Put input and output quantities into matrices X and Y
X=as.matrix(cbind(x1,x2))
Y=as.matrix(y1)

#set up output orientation model for DMU 1 and VRS
objtype='max'
obj=c(rep(0,nDMU),1)
A=matrix(c(rep(1,nDMU)+1),1)
A[1,1:nDMU]=t(Y); A[1,(nDMU+1)]=-Y[1,1]
A[2:nDMU,1:nDMU]=t(X)
A[4,1:nDMU]=1
b=c(0,X[1,1],1)
rest=c(>=,'','',<=,'','=','')

#set up lpSolveAPI object
LP_API=make.lp(nrow=A,ncol=ncol(A))
lp.control(LP_API,sense=objtype)
set.objfn(LP_API,obj)
for(i in 1:nrow(A)){
  set.row(LP_API,1,A[i,])
}
set.constr.type(LP_API,rest)
set.rhs(LP_API,b)
#solve with lpSolveAPI
#solve=lpSolveAPI()
#solve

tmp2=mmlp(objtype=obj,rest=rest,rhs=b)
tmp2$objval
#solve efficiency scores for all DMU's
time_API=0;time_MMLP=0
effhat_API=0;effhat_MMLP=0
#i=1
for(i in 1:nDMU){
  #
normgen

generates normal random variates using uniform variates generated from Fortran code: mt19937.f90

Description

generates normal random variates using the uniform twister algorithm in Fortran code: mt19937.f90
Usage

\texttt{normgen(n, seedval)}

Arguments

- \texttt{n} \hspace{1cm} \text{number of random numbers to simulate}
- \texttt{seedval} \hspace{1cm} \text{a positive 32 bit integer seedvalue}

Value

Returns a vector of normal variates

Author(s)

R and Fortran interface to mt19937.f90 code: Joe Atwood

Examples

\begin{verbatim}
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
x=normgen(100, seedval=2014)
\end{verbatim}

\texttt{sampleum} \quad \textit{Samples with replacement using Fortran generated random numbers.}

Description

Samples with replacement using Fortran generated random numbers.

Usage

\texttt{sampleum(nobs, x, seedval)}

Arguments

- \texttt{nobs} \hspace{1cm} \text{sample size}
- \texttt{x} \hspace{1cm} \text{values to sample from}
- \texttt{seedval} \hspace{1cm} \text{seed value for fortran random number generator}

Value

\texttt{xsample} \hspace{1cm} \text{nobs values resampled from x}

Author(s)

Joe Atwood <jatwood@montana.edu>

Examples

\texttt{sampleum(10, 1:5)}
**seconds**

Pulls seconds from system clock.

**Description**

Pulls seconds from system clock.

**Usage**

seconds()

**Arguments**

none

**Value**

seconds from system clock

**Author(s)**

Joe Atwood <jatwood@montana.edu>

**Examples**

seconds()

---

**ugen**

obtains uniform random variates using Fortran code: mt19937.f90

**Description**

generates uniform random variates using the twister algorithm in Fortran code: mt19937.f90

**Usage**

ugen(n,seedval)

**Arguments**

- **n**: number of random numbers to simulate
- **seedval**: a positive 32 bit integer seedvalue

**Value**

Returns a vector of uniform variates

**Author(s)**

R and Fortran interface to mt19937.f90 code: Joe Atwood
Examples

\[ x = \text{ugen}(100, \text{seedval}=2014) \]
Index

de.a. 3
dea.boot DEAb loot, 3
dea.sample.DEAb loot, 4
DEAb loot, 5
DEAb loot-package, 1

MMLP, 8

normgen, 13

sampleum, 14
seconds, 15

ugen, 15