Package 'turboEM'

January 13, 2020

Title A Suite of Convergence Acceleration Schemes for EM, MM and Other Fixed-Point Algorithms

Description Algorithms for accelerating the convergence of slow, monotone sequences from smooth, contraction mapping such as the EM and MM algorithms. It can be used to accelerate any smooth, linearly convergent acceleration scheme. A tutorial style introduction to this package is available in a vignette on the CRAN download page or, when the package is loaded in an R session, with vignette(``turboEM'').

Depends R (>= 2.12.0), doParallel, foreach, numDeriv, quantreg

Imports iterators

Suggests setRNG

Version 2020.1

LazyLoad yes

License GPL-2

Author Jennifer F. Bobb [aut],
Ravi Varadhan [aut, cre],
Hui Zhao [ctb]

Maintainer Ravi Varadhan <ravi.varadhan@jhu.edu>

URL http://www.jhsph.edu/agingandhealth/People/Faculty_personal_pages/Varadhan.html

Repository CRAN

NeedsCompilation no

Date/Publication 2020-01-13 16:40:02 UTC

R topics documented:

`implants` *Fetal Death in Mice*

Description

Data on the number of fetal deaths arising from dominant lethal testing in mice.

Usage

```
data(implants)
```

Format

A data frame containing the number of dead and survived implants from 523 mice.

Source

Zhou H and Lange K (2010). MM Algorithms for Some Discrete Multivariate Distributions. *Journal of Computational and Graphical Statistics*. 19 (3) 645-665. Supplementary material.

References

Haseman JK and Soares ER (1976). The Distribution of Fetal Death in Control Mice and Its Implications on Statistical Tests for Dominant Lethal Effects, *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*. 41, 277-288.

`parties` *Political Parties*

Description

Political parties of members of the U.S. House of Representatives, 2005.

Usage

```
data(votes)
```

Format

A vector of integers representing the political parties, with 0, 1, and 2 corresponding to Republicans, Democrats, and Independents, respectively.

Source

Diaconis PD, Goel S, and Holmes S (2008). Horseshoes in Multidimensional Scaling and Local Kernel Methods. *Annals of Applied Statistics*. 2 (3) 777-807. Supplementary material.

---

psychfactors *Psychiatric Test Correlations*

Description

Intercorrelations of outcomes from a set of psychiatric tests for 148 children.

Usage

```
data(psychfactors)
```

Format

A 10-by-10 correlation matrix.

Source

Maxwell AE (1961). Recent Trends in Factor Analysis. *Journal of the Royal Statistical Society. Series A (General)*. 124 (1) 49-59.

---

rats *Population Growth of Rats*

Description

Longitudinal measurements of the weights of rats in control and treatment groups.

Usage

```
data(rats)
```

Format

A data frame containing weights from 60 rats divided into two groups, with each rat measured at 5 time points.

Source

Gelfand AE, Hills SE, Racine-Poon A, and Smith AFM (1990). Illustration of Bayesian inference in normal data models using Gibbs sampling. *Journal of the American Statistical Association*. 85, 972-985.

---

turbo *Methods for objects of class "turbo"*

Description

The turbo class represents results from parameter estimation in fixed-point mapping problems. The turboem function outputs objects of class turbo.

Usage

```
## S3 method for class 'turbo'
print(x, ...)
## S3 method for class 'turbo'
pars(x, ...)
## S3 method for class 'turbo'
error(x, ...)
## S3 method for class 'turbo'
plot(x, which.methods = seq_along(x$method),
method.names = x$method[which.methods], xlim, ylim, ...)
## S3 method for class 'turbo'
grad(x, objfn=x$objfn, which.methods = seq_along(x$method),
  method.names = x$method[which.methods], ...)
## S3 method for class 'turbo'
hessian(x, objfn=x$objfn, which.methods = seq_along(x$method),
  method.names = x$method[which.methods], ...)
## S3 method for class 'turbo'
stderror(x, objfn=x$objfn, which.methods = seq_along(x$method),
  method.names = x$method[which.methods], ...)
```

Arguments

x	An object of class turbo, typically the output of a call to turboem.
which.methods	A vector identifying for which subset of algorithms results are desired.
method.names	A vector of unique identifiers for the algorithms for which results are being provided.
xlim	Optional range for the x-axis of the trace plot.
ylim	Optional range for the y-axis of the trace plot.
objfn	Objective function. Usually this is taken to be the appropriate component of a turbo object.
...	Additional arguments.

Value

print	Shows a brief summary of the results from fitting the acceleration schemes.
pars	Prints the fixed-point values across acceleration schemes at termination of the algorithms.
error	Prints any error messages from running the acceleration schemes
plot	Shows a trace plot of the objective function value over iterations. This method is only available if the call to turboem had the argumentcontrol.run[["keep.objfval"]]=TRUE
grad	Calculates the gradient of the objective function evaluated at the fixed-point across acceleration schemes. Uses numerical methods from the package numDeriv.
hessian	Calculates the Hessian of the objective function evaluated at the fixed-point across acceleration schemes. Uses numerical methods from the package numDeriv.
stderror	Provides estimates of the standard error of the fixed-point across acceleration schemes.

See Also

turboem

Examples

```
##############################################################################
# Also see the vignette by typing:
#  vignette("turboEM")
#
# EM algorithm for Poisson mixture estimation

fixptfn <- function(p,y) {
# The fixed point mapping giving a single E and M step of the EM algorithm
#
pnew <- rep(NA,3)
i <- 0:(length(y)-1)
zi <- p[1]*exp(-p[2])*p[2]^i / (p[1]*exp(-p[2])*p[2]^i + (1 - p[1])*exp(-p[3])*p[3]^i)
pnew[1] <- sum(y*zi)/sum(y)
pnew[2] <- sum(y*i*zi)/sum(y*zi)
pnew[3] <- sum(y*i*(1-zi))/sum(y*(1-zi))
p <- pnew
return(pnew)
}

objfn <- function(p,y) {
# Objective function whose local minimum is a fixed point
# negative log-likelihood of binary poisson mixture
i <- 0:(length(y)-1)
loglik <- y*log(p[1]*exp(-p[2])*p[2]^i/exp(lgamma(i+1)) +
(1 - p[1])*exp(-p[3])*p[3]^i/exp(lgamma(i+1)))
return ( -sum(loglik) )
}

# Real data from Hasselblad (JASA 1969)
```

```
poissmix.dat <- data.frame(death=0:9, freq=c(162,267,271,185,111,61,27,8,3,1))
y <- poissmix.dat$freq

# Use a preset seed so the example is reproducable.
require("setRNG")
old.seed <- setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion",
    seed=1))

p0 <- c(runif(1),runif(2,0,4))  # random starting value

# Basic EM algorithm, SQUAREM, and parabolic EM, with default settings
res1 <- turboem(par=p0, y=y, fixptfn=fixptfn, objfn=objfn, method=c("EM", "squarem", "pem"))

# Apply methods for class "turbo"
res1
pars(res1)
grad(res1)
hessian(res1)
stderror(res1)
error(res1)

# We get an error for Dynamic ECME (decme) if we do not specify the boundary function
res2 <- turboem(par=p0, y=y, fixptfn=fixptfn, objfn=objfn,
  method=c("EM", "squarem", "pem", "decme"))
res2
error(res2)

# we can't plot the results, because we did not store the objective function value at each iteration
# Changing the options to store the objective function values, we can:
res1keep <- turboem(par=p0, y=y, fixptfn=fixptfn, objfn=objfn, method=c("EM", "squarem", "pem"),
  control.run=list(keep.objfval=TRUE))
plot(res1keep, xlim=c(0.001, 0.02))
```

---

turboem *A suite of acceleration schemes for fixed-point iterations*

Description

Globally-convergent, partially monotone, acceleration schemes for accelerating the convergence of *any* smooth, monotone, slowly-converging contraction mapping. It can be used to accelerate the convergence of a wide variety of iterations including the expectation-maximization (EM) algorithms and its variants, majorization-minimization (MM) algorithm, power method for dominant eigenvalue-eigenvector, Google's page-rank algorithm, and multi-dimensional scaling.

Usage

```
turboem(par, fixptfn, objfn, method = c("em","squarem","pem","decme","qn"),
boundary, pconstr = NULL, project = NULL, parallel = FALSE, ...,
  control.method = replicate(length(method),list()), control.run = list())
```

Arguments

`par`	A vector of parameters denoting the initial guess for the fixed point.
`fixptfn`	A vector function, F that denotes the fixed-point mapping. This function is the most essential input in the package. It should accept a parameter vector as input and should return a parameter vector of same length. This function defines the fixed-point iteration: $x_{k+1} = F(x_k)$. In the case of EM algorithm, F defines a single E and M step.
`objfn`	This is a scalar function, L, that denotes a "merit" function which attains its local minimum at the fixed-point of F. This function should accept a parameter vector as input and should return a scalar value. In the EM algorithm, the merit function L is the negative log-likelihood. In some problems, a natural merit function may not exist. However, this argument is required for all of the algorithms *except* Squarem (which defaults to Squarem-2 if `objfn` not provided) and EM.
`method`	Specifies which algorithm(s) will be applied. Must be a vector containing one or more of `c("em","squarem","pem","decme","qn")`.
`boundary`	Argument required for Dynamic ECME (`decme`) only. Function to define the subspaces over which the line search is conducted.
`pconstr`	Optional function for defining boundary constraints on parameter values. Function maps a vector of parameter values to TRUE if constraints are satisfied. Note that this argument is only used for the Squarem (`squarem`), Parabolic EM (`pem`), and quasi-Newton (`qn`) algorithms, and it has no effect on the other algorithms.
`project`	Optional function for defining a projection that maps an out-of-bound parameter value into the constrained parameter space. Requires the `pconstr` argument to be specified in order for the `project` to be applied.
`parallel`	Logical indicating whether the acceleration schemes will be run in parallel. Note that the parallel implementation is based on the `foreach` package, which depends on a *parallel backend* being registered prior to running `turboem()`. See *Details* of `foreach`.
`control.method`	If `method = c(method1,method2,...)`, then `control.method = list(list1,list2,...)` where `list1` is the list of control parameters for `method1`, `list2` is the list of control parameters for `method2`, and so on. If `length(method) == 1`, then `control.method` is the list of control parameters for the acceleration scheme. See *Details*.
`control.run`	List of control parameters for convergence and stopping the algorithms. See *Details*.
`...`	Arguments passed to `fixptfn` and `objfn`.

Details

The function `turboem` is a general-purpose algorithm for accelerating the convergence of any slowly-convergent (smooth) fixed-point iteration.

The component lists of the `control.method` are used to specify any changes to default values of algorithm control parameters. Full names of control list elements must be specified, otherwise, user specifications are ignored. Default control parameters for `method="squarem"` are `K=1`, `square=TRUE`, `version=3`, `step.min0=1`, `step.max0=1`, `mstep=4`, `kr=1`, `objfn.inc=1`. Default

control parameters for method="pem" are l=10, h=0.1, a=1.5, and version="geometric". Default control parameters for method="decme" are version="v2" and tol_op=0.01. Default control parameters for method="qn" are qn=5.

Default values of control.run are: convtype = "parameter", tol = 1.0e-07, stoptype = "maxiter", maxiter = 1500, maxtime = 60, convfn.user = NULL, stopfn.user = NULL, trace = FALSE, keep.objfval = FALSE.

There are two ways the algorithm will terminate. Either the algorithm will terminate if convergence has been achieved, or the algorithm will terminate if convergence has not been achieved within a pre-specified maximum number of iterations or maximum running time. The arguments convtype, tol, and convfn.user control the convergence criterion. The arguments stoptype, maxiter, maxtime, and stopfn.user control the alternative stopping criterion.

Two types of convergence criteria have been implemented, with an option for the user to define his/her own convergence criterion. If convtype = "parameter", then the default convergence criterion is to terminate if sqrt(crossprod(new -old)) < tol, where new denotes the current value of the fixed point and old denotes the previous fixed-point value. If convtype = "objfn", then the default convergence criterion is to terminate if abs(new -old) < tol, where new denotes the current value of the objective function and old denotes the previous value of the objective function. If the user desires alternate convergence criteria, convfn.user may be specified as a function with inputs new and old that maps to a logical taking the value TRUE if convergence is achieved and the value FALSE if convergence is not achieved.

Two types of alternative stopping criteria have been implemented, with the option for the user to define his/her own stopping criterion. If stoptype = "maxiter", then the algorithm will terminate if convergence has not been achieved within maxiter iterations of the acceleration scheme. If stoptype = "maxtime", then the algorithm will terminate if convergence has not been achieved within maxtime seconds of running the acceleration scheme. Note: the running time of the acceleration scheme is calculated once every iteration. If the user desires different alternate stopping criteria than those implemented, stopfn.user may be specified as a function with no inputs that maps to a logical taking the value TRUE which leads to the algorithm being terminated or the value FALSE which leads to the algorithm proceeding as usual.

convtype A character string equal to "parameter" or "objfn". "parameter" indicates that the convergence criterion is a function of the current and previous value of the fixed point. objfn indicates that the convergence criterion is a function of the current and previous value of the objective function.

tol A small, positive scalar that determines when convergence is achieved. See details above for convergence criteria currently implemented. Default is 1.e-07.

stoptype A character string equal to "maxiter" or "maxtime" that determines an alternative stopping rule for the algorithm. See details above for stopping rules currently implemented. Default is "maxiter".

maxiter If stoptype = "maxiter", specifies the number of iterations after which the algorithm will be terminated if convergence has not been achieved. Default is 1500.

maxtime If stoptype = "maxtime", specifies the running time (in seconds) after which the algorithm will be terminated if convergence has not been achieved. Default is 60.

convfn.user Optional, user-specified function for determining whether convergence has been achieved. Function should take as inputs new and old, where new is the current value (of the fixed point if convtype = "parameter" and of the objective function value if convtype

= "objfn") and old is the previous value. Function should map to a logical taking the value TRUE if convergence is achieved (and hence the algorithm is terminated) and the value FALSE if convergence is not achieved. Default is NULL.

stopfn.user Optional, user-specified function for determining whether to terminate the algorithm if convergence has not been achieved. See details above for how to specify. Default is NULL.

trace A logical variable denoting whether some of the intermediate results of iterations should be displayed to the user. Default is FALSE.

keep.objfval A logical variable denoting whether the objective function value at each iteration should be stored. Default is FALSE.

Value

turboem returns an object of class turbo. An object of class turbo is a list containing at least the following components:

fail	Vector of logical values whose jth element indicates whether algorithm j failed (produced an error)
value.objfn	Vector of the value of the objective function L at termination for each algorithm.
itr	Vector of the number of iterations completed for each algorithm.
fpeval	Vector of the number of fixed-point evaluations completed for each algorithm.
objfeval	Vector of the number of objective function evaluations completed for each algorithm.
convergence	Vector of logical values whose jth element indicates whether algorithm j satisfied the convergence criterion before termination
runtime	Matrix whose jth row contains the "user", "system", and "elapsed" time for running the jth algorithm.
errors	Vector whose jth element is either NA or contains the error message from running the jth algorithm
pars	Matrix whose jth row contains the fixed-point parameter values at termination for the jth algorithm.
trace.objfval	If control.run[["keep.objfval"]]=TRUE, contains a list whose jth component is a vector of objective function values across iterations for the jth algorithm

References

R Varadhan and C Roland (2008). Simple and globally convergent numerical schemes for accelerating the convergence of any EM algorithm. *Scandinavian Journal of Statistics*, 35:335-353.

A Berlinet and C Roland (2009). Parabolic acceleration of the EM algorithm. *Stat Comput*. 19 (1) 35-47.

Y He and C Liu (2010) The Dynamic ECME Algorithm. Technical Report. arXiv:1004.0524v1.

H Zhou, DH Alexander, and KL Lange (2011). A quasi-Newton acceleration for high-dimensional optimization algorithms. *Stat Comput*. 21 (2) 261-273.

See Also

turbo

Examples

```
##############################################################################
# Also see the vignette by typing:
#  vignette("turboEM")
#
# EM algorithm for Poisson mixture estimation

fixptfn <- function(p,y) {
# The fixed point mapping giving a single E and M step of the EM algorithm
#
pnew <- rep(NA,3)
i <- 0:(length(y)-1)
zi <- p[1]*exp(-p[2])*p[2]^i / (p[1]*exp(-p[2])*p[2]^i + (1 - p[1])*exp(-p[3])*p[3]^i)
pnew[1] <- sum(y*zi)/sum(y)
pnew[2] <- sum(y*i*zi)/sum(y*zi)
pnew[3] <- sum(y*i*(1-zi))/sum(y*(1-zi))
p <- pnew
return(pnew)
}

objfn <- function(p,y) {
# Objective function whose local minimum is a fixed point
# negative log-likelihood of binary poisson mixture
i <- 0:(length(y)-1)
loglik <- y*log(p[1]*exp(-p[2])*p[2]^i/exp(lgamma(i+1)) +
(1 - p[1])*exp(-p[3])*p[3]^i/exp(lgamma(i+1)))
return ( -sum(loglik) )
}

# Real data from Hasselblad (JASA 1969)
poissmix.dat <- data.frame(death = 0:9,
freq = c(162,267,271,185,111,61,27,8,3,1))
y <- poissmix.dat$freq

# Use a preset seed so the example is reproducable.
require("setRNG")
old.seed <- setRNG(list(kind = "Mersenne-Twister", normal.kind = "Inversion",
    seed = 54321))

p0 <- c(runif(1),runif(2,0,4))  # random starting value

# Basic EM algorithm, SQUAREM, and parabolic EM, with default settings
res1 <- turboem(par = p0, y = y, fixptfn = fixptfn, objfn = objfn,
method = c("EM", "squarem", "pem"))

# To apply the dynamic ECME (decme) acceleration scheme,
# we need to include a boundary function
boundary <- function(par, dr) {
```

```
lower <- c(0, 0, 0)
upper <- c(1, 10000, 10000)
low1 <- max(pmin((lower-par)/dr, (upper-par)/dr))
upp1 <- min(pmax((lower-par)/dr, (upper-par)/dr))
return(c(low1, upp1))
}
res2 <- turboem(par = p0, y = y, fixptfn = fixptfn, objfn = objfn,
boundary = boundary, method = c("EM", "squarem", "pem", "decme"))

# change some of the algorithm-specific default specifications (control.method),
# as well as the global control parameters (control.run)
res3 <- turboem(par = p0, y = y, fixptfn = fixptfn, objfn = objfn,
  boundary = boundary, method = c("em", "squarem", "squarem", "decme", "qn", "qn"),
control.method = list(list(), list(K = 2), list(K = 3),
list(version = "v3"), list(qn = 1), list(qn = 2)),
control.run = list(tol = 1e-12, stoptype = "maxtime", maxtime = 1))

# Only the standard EM algorithm and SQUAREM *do not* require
# providing the objective function.
res4 <- turboem(par = p0, y = y, fixptfn = fixptfn,
method = c("em", "squarem", "squarem"),
control.method = list(list(), list(K = 1), list(K = 2)))
# If no objective function is provided, the "squarem" method defaults to Squarem-2
# Or, if control parameter K > 1, it defaults to Cyclem-2.
# Compare Squarem with and without objective function provided:
res5 <- turboem(par = p0, y = y, fixptfn = fixptfn, method = "squarem")
res5
res6 <- turboem(par = p0, y = y, fixptfn = fixptfn, objfn = objfn, method = "squarem")
res6
```

`turboSim` *Conduct benchmark studies of EM accelerator*

Description

The `turboSim` function conducts benchmark studies to compare performance of multiple acceleration schemes over a large number of repetitions. The `turboSim` function outputs objects of class `turbosim`.

Usage

```
turboSim(parmat, fixptfn, objfn, method = c("em","squarem","pem","decme","qn"),
  boundary, pconstr = NULL, project = NULL, parallel = FALSE, method.names,
  keep.pars = FALSE, ..., control.method = replicate(length(method),list()),
  control.run = list())
```

Arguments

parmat	A matrix of starting parameter values, where each row corresponds to a single benchmark study repetition.
fixptfn	A vector function, F that denotes the fixed-point mapping. This function is the most essential input in the package. It should accept a parameter vector as input and should return a parameter vector of same length. This function defines the fixed-point iteration: $x_{k+1} = F(x_k)$. In the case of EM algorithm, F defines a single E and M step.
objfn	This is a scalar function, L, that denotes a "merit" function which attains its local minimum at the fixed-point of F. This function should accept a parameter vector as input and should return a scalar value. In the EM algorithm, the merit function L is the negative log-likelihood. In some problems, a natural merit function may not exist. However, this argument is required for all of the algorithms *except* Squarem (which defaults to Squarem-2 if objfn not provided) and EM.
method	Specifies which algorithm(s) will be applied. Must be a vector containing one or more of c("em","squarem","pem","decme","qn").
boundary	Argument required for Dynamic ECME (decme) only. Function to define the subspaces over which the line search is conducted.
pconstr	Optional function for defining boundary constraints on parameter values. Function maps a vector of parameter values to TRUE if constraints are satisfied. Note that this argument is only used for the Squarem (squarem), Parabolic EM (pem), and quasi-Newton (qn) algorithms, and it has no effect on the other algorithms.
project	Optional function for defining a projection that maps an out-of-bound parameter value into the constrained parameter space. Requires the pconstr argument to be specified in order for the project to be applied.
parallel	Logical indicating whether the *repetitions* of the benchmark study will be run in parallel. Note that the parallel implementation is based on the foreach package, which depends on a *parallel backend* being registered prior to running turboSim(). See *Details* of foreach.
method.names	Vector of unique names that identify the algorithms being compared.
keep.pars	Logical indicating whether the parameter values at termination should be kept. Defaults to FALSE.
control.method	If method = c(method1,method2,...), then control.method = list(list1,list2,...) where list1 is the list of control parameters for method1, list2 is the list of control parameters for method2, and so on. If length(method) == 1, then control.method is the list of control parameters for the acceleration scheme. See *Details* of turboem.
control.run	List of control parameters for convergence and stopping the algorithms. See *Details* of turboem.
...	Arguments passed to fixptfn and objfn.

Value

turboSim returns an object of class turbosim.

See Also

turbosim, turboem

Examples

```
##########################################################################
# Examples provided in the vignette, which can be seen by typing
#  vignette("turboEM")
```

---

turbosim *Methods for objects of class "turbosim"*

Description

The turbosim class represents results from benchmark studies of algorithms to acceleration parameter estimation in fixed-point mapping problems.

Usage

```
## S3 method for class 'turbosim'
print(x, ...)
## S3 method for class 'turbosim'
summary(object, which.methods = seq_along(object$method),
method.names = object$method.names[which.methods], eps = 0.1, sol = NULL, ...)
## S3 method for class 'turbosim'
boxplot(x, which.methods = seq_along(x$method),
method.names = x$method.names[which.methods],
whichfail = (x$fail | !x$conv)[,which.methods], xunit="sec", log=FALSE, ...)
## S3 method for class 'turbosim'
dataprof(x, which.methods = seq_along(x$method),
method.names = x$method.names[which.methods],
whichfail = (x$fail | !x$conv)[,which.methods], col, lty, nout = 50, xlim, ...)
## S3 method for class 'turbosim'
pairs(x, which.methods=seq_along(x$method),
method.names = x$method.names[which.methods],
whichfail = (x$fail | !x$conv)[,which.methods], ...)
```

Arguments

object	An object of class turbosim, the structure of which is described in *Details*.
x	An object of class turbosim, the structure of which is described in *Details*.
which.methods	A vector identifying for which subset of algorithms results are desired.
method.names	A vector of unique identifiers for the algorithms for which results are being provided.

`eps`	Used to define a tolerance between the objective function value attained by a particular acceleration scheme and the best achievable objective function value (either across schemes or as defined by the user). See *Details*.
`sol`	Optional argument defining the best achievable objective function value for a given fixed-point mapping problem. Defaults to NULL. See *Details*.
`xunit`	Units for running time to be used in the boxplots. Argument takes the value "sec" or "min."
`log`	Logical indicating whether the log of the running time will be plotted. Defaults to FALSE.
`whichfail`	A matrix of logical values where the (i,j)-entry indicates whether algorithm j of simulation iteration i failed (however the user wishes to define a failure for visualization purposes). If argument is not provided by user, then by default a failure is defined to be the event where the algorithm produces an error *or* does not converge.
`col`	Optional argument: A vector where each component defines the color for the line corresponding to each algorithm being compared.
`lty`	Optional argument: A vector where each component defines the line-type for the line corresponding to each algorithm being compared.
`nout`	Number of values at which the empirical distribution function is estimated. Should be less than the number of simulation iterations.
`xlim`	Optional argument: Defines the x-axis limits for the data profile. Defaults to the full range of the running times over all algorithms being plotted.
`...`	Additional arguments.

Details

An object of class `turbosim` is typically the product of the function `turboSim`. It is a list containing at least the following components:

`method.names` Vector of unique identifiers for the algorithms being compared

`fail` Matrix whose (i,j)-element is a logical (TRUE/FALSE) for whether the jth algorithm at the ith benchmark study repetition failed (produced an error).

`convergence` Matrix whose (i,j)-element is a logical (TRUE/FALSE) for whether the jth algorithm at the ith benchmark study repetition satisfied the convergence criterion before termination.

`value.objfn` Matrix whose (i,j)-element is the value of the objective function of the jth algorithm at the ith benchmark study repetition.

`runtime` Matrix whose (i,j)-element is the running time of the jth algorithm at the ith benchmark study repetition.

`itr` Matrix whose (i,j)-element is the number of completed iterations of the jth algorithm at the ith benchmark study repetition.

`fpeval` Matrix whose (i,j)-element is the number of fixed-point function evaluations of the jth algorithm at the ith benchmark study repetition.

`objfeval` Matrix whose (i,j)-element is the number of objective function evaluations of the jth algorithm at the ith benchmark study repetition.

`errors` Matrix whose (i,j)-element contains the error message produced by the jth algorithm at the ith benchmark study repetition (if there was an error).

This list usually will also contain the components `fixptfn`, `objfn`, `method`, `pconstr`, `project`, `control.method`, and `control.run`, which were provided as arguments for `turboSim`.

The `summary` function shows a table of the number of failures across acceleration schemes. There are three types of failures. The first occurs when the algorithm produces an error message. The second is if the algorithm does not converge before the alternative stopping rule is achieved (e.g. the maximum number of iterations or maximum pre-specified runtime is reached). The third is if the algorithm claims convergence but the value of the objective function is "far" from the best achievable value. To assess this third type of failure, we determine whether the objective function value achieved by the algorithm is close (within `eps`) to the smallest value achieved across all algorithms at that simulation iteration. Alternatively, if the user knows a priori the true objective function value, he/she may specify the argument `sol`, in which case, the third type of failure occurs when the objective function value achieved by the algorithm is within `eps` of `sol`.

Further details for each of the methods are provided in the vignette, which can be seen by typing `vignette("turboEM")`.

Value

`summary`	Summarizes the number of failures by type across simulation iterations for each acceleration scheme.
`boxplot`	Shows box plots of algorithm running times for each acceleration scheme.
`dataprof`	Plots the data profile, or the estimated distribution function of the time until convergence for each acceleration scheme.
`pairs`	Scatterplot matrix showing pairwise comparison of the running times for each pair of acceleration schemes.

See Also

`turboem`, `turbo`

Examples

```
##############################################################################
# Examples provided in the vignette, which can be seen by typing
#  vignette("turboEM")
```

votes *Roll Call Votes*

Description

Roll call votes from the U.S. House of Representatives, 2005.

Usage

```
data(votes)
```

Format

A 401-by-669 matrix whose (i,j)-entry corresponds to the vote of the ith representative on the jth roll call. Possible votes are "yea", "nay", or "not voting", which are represented by 1/2, -1/2, and 0, respectively.

Source

Diaconis PD, Goel S, and Holmes S (2008). Horseshoes in Multidimensional Scaling and Local Kernel Methods. *Annals of Applied Statistics*. 2 (3) 777-807. Supplementary material.

Index