

Package ‘accuracy’

January 22, 2010

Title Tools for testing and improving accuracy of statistical results.

Version 1.23

VersionSplus 1.23-2

Date 2007-07-25

Author Micah Altman, Jeff Gill, Michael P. McDonald

Description This is a suite of tools designed to test and improve the accuracy of statistical computation, including: Summarization of the sensitivity of linear and non-linear models (lm, glm, mle, nls) to measurement and numerical error; Sensitivity analysis of dozens of models as run through Zelig; A generalized cholesky method for correcting non-invertable Hessians; Tests for the global optimality of non-linear regression and maximum likelihood results; Tools for obtaining true random numbers using entropy collected from the system and/or entropy servers on the internet; A method for converting floating point numbers to normalized fractions; Benchmark data for checking the accuracy of basic distribution functions.

Maintainer Micah Altman <Micah_Altman@harvard.edu>

MaintainerSplus Stephen Kaluzny <skaluzny@tibco.com>

Headshot http://www.hmdc.harvard.edu/micah_altman/micah_altman.jpg

Suggests Zelig

SuggestsSplus

Imports methods

ImportsSplus

License GPL 2.0

URL <http://www.r-project.org>,
http://www.hmdc.harvard.edu/micah_altman/numal/

R topics documented:

dehaan	2
Distribution Test Data, and log relative error function	3
frexp	5
HTML.sensitivity.summary	6
modelsCompare	8
plot.sensitivity	10
psim	11
PTBdefault	12
PTBdiscrete	13
PTBi	14
PTBmu.gen	17
reclass.mat.diag	18
runifS	19
sechol	21
sensitivityZelig	23
sensitivity	25
starr	29

dehaan

dehaan global optimality test

Description

Implements the de Haan test for identification of the global optimum of a likelihood surface.

Usage

```
dehaan(l1Test, l1Max, pval=.05 )
```

Arguments

<code>l1Test</code>	Vector of randomly generated likelihood values
<code>l1Max</code>	Value of the likelihood function for a candidate global optimum
<code>pval</code>	p-value of the test

Details

`\texttt{dehaan}` computes a (1-p) confidence interval for the global optimum of a likelihood surface from a vector of user-supplied randomly chosen likelihood values. A user-supplied candidate maximum likelihood value is evaluated against confidence interval.

Value

`\texttt{dehaan}=TRUE` if the candidate value is greater than the (1-p) confidence interval for the true optimum.

Note

The choice of the vector of user-supplied random likelihood values is somewhat of an art. The parameter space should be large enough to capture the true optimum, but not so large as to include illegal or impractical parameter values.

The generated confidence interval depends on the number of random draws from the parameter space. The number should be large enough to allow the application of asymptotic theory. A recommended number of random evaluations of the likelihood function at a given set of parameter values is 500 or more, but depends on the size of the parameter space. Users are encouraged to experiment with more and less draws and observe results.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/, Michael P. McDonald

References

- Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/
- de Haan, L. 1981. "Estimation of the Minimum of a Function Using Order Statistics." *Journal of the American Statistical Association* 76, 467-9.
- Veall, M. R. 1990. "Testing for a Global Maximum in an Econometric Context." *Econometrica* 58 1459-65.

Examples

```
# The deHaan test is constructed as a maximum likelihood
# test, with negative values for the likelihood. The BOD problem
# is a non-linear least squares minimization problem. This test
# is implemented using the negative of the sum of squares for consistency
# with the deHaan framework of maximum likelihood.

BOD <-
structure(list(Time = c(1, 2, 3, 4, 5, 7), demand = c(8.3, 10.3,
19, 16, 15.6, 19.8)), .Names = c("Time", "demand"), row.names = c("1",
"2", "3", "4", "5", "6"), class = "data.frame", reference = "A1.4, p. 270")
stval<-expand.grid(A = seq(10, 100, 10), lrc = seq(.5, .8, .1))
llfun<-function(A,lrc,BOD)
  -sum((BOD$demand - A*(1-exp(-exp(lrc)*BOD$Time)))^2)
lls<-NULL
for (i in 1:nrow(stval)) {
  lls = rbind(lls, llfun(stval[i,1], stval[i,2],BOD))
}
fm1 <- nls(demand ~ A*(1-exp(-exp(lrc)*Time)),
  data = BOD, start = c(A = 20, lrc = log(.35)))
ss = -sum(resid(fm1)^2)
dehaan(lls, ss)
```

Distribution Test Data, and log relative error function

Benchmark data to test the accuracy of statistical distribution functions, function to compute log relative error

Description

This is benchmark data, used to test the accuracy of statistical distribution functions. Also included is a function for computing log-relative error, a measure of accuracy.

Usage

```
data(ttst)
data(ftst)
data(gammatst)
data(normtst)
data(chisqtst)
```

Details

These tests values are tab-delimited ascii datasets, with a variable header line. Typical variables are:

P - probability value DF - degrees of freedom INVDIST - inv the inverse function value for P: i.e.: $\text{invt}(P,DF) = \text{INVT}$ INVDIST - inv the inverse function value for P: i.e.: $\text{invt}(P,DF) = \text{INVT}$ PINVDIST - value for cumulative distribution of INVDIST i.e.: $\text{cumt}(\text{INVT},DF) = \text{PINVT}$

Note that because of limits in numerical precision, $\text{cumt}(\text{invt}(P,DF))$ is not always equal to P

Of course, not all possible values of P and DF can be listed. The test values for P were created from systematic and random samples in $[1e-12,1-1e-12], 0,1$, and $[0,1E5]$ respectively.

We use Kneusel's ELV program (1989) to calculate the values the cumulative and inverse distributions. The results are claimed by Kneusel to be accurate to 6 digits. We checked these results using Brown's (1998) DCDFLIB library. The results of both calculations agreed to the 6 digits supplied by ELV, but in a small number of cases, DCDFSTAT's calculation at the 7th digit indicated that the sixth digit would change if rounded.

Note that both ELV and DCDFLIB can generate many more distributions than are included here. Other resources are described in the references

Value

Returns a new vector of log-relative-errors (log absolute error where $\$c_i\$ == 0$). The resulting values are roughly interpretable as the number of significant digits of agreement between c and x. Larger numbers indicate that x is a relatively more accurate approximation of c. Numbers less than or equal to zero indicate complete disagreement between x and c.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmhc.harvard.edu/micah_altman/, Michael McDonald

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmhc.harvard.edu/numerical_issues/

Altman, Micah and Michael McDonald, 2001. "Choosing Reliable Statistical Software." *PS: Political Science & Politics* 24(3): 681-8

Barry W. Brown, James Lovato, Kathy Russell, 1998, DCDFLIB, <ftp://odin.mda.uth.tmc.edu/pub/unix/dcdfplib.c-1.0-tar.Z>

Kneusel, L. 1989. *Computergesteuerte Berechnung statistischer Verteilungen*. Olde nbourg, Meunchen-Wien. <http://www.stat.uni-muenchen.de/~kneusel/elv/elv.html>

Examples

```
# simple LRE examples
LRE(1.001,1) # roughly 3 significant digits agreement
LRE(1,1) # complete agreement
LRE(20,1) # complete disagreement

#
# how accurate are student s t-test functions?
#

data(ttst)
# compute t quantiles using benchmark data
tqt = qt(ttst$p,ttst$df)

# compute log-relative-error (LRE) of qt() results, compared to
# correct answers

lrq = LRE(tqt, ttst$inv);

# if there are entries with LRE s of < 5, there may be
# significant inaccuracies in the qt() function

table(trunc(lrq))

# now repeat process, for pt()

tpt = pt(ttst$inv,ttst$df)
lrp= LRE(tpt, ttst$pinv);
table(trunc(lrp))
```

frexp	<i>Function to convert vector of floating-point numbers to fractional and integral components</i>
--------------	---

Description

The `frexp()` function returns a matrix with the normalized fraction $[-.5,1)$ of the vector in the first column and the exponent (a power of two) in the second column. (If x is zero, then the normalized fraction is zero and zero is stored in the exponent.)

Usage

```
frexp(v)
```

Arguments

`v` vector of doubles

Details

This is an R wrapper around the `<math.h>` `frexp()` function.

Value

Returns a $2 \times n$ matrix. The normalized fraction $[-.5,1)$ of x is the first column and the exponent (a power of two) in the second column. (If $x[i]$ is zero, then the normalized fraction of $x[i]$ is zero and zero is stored in the exponent.)

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

Examples

```
x = runif(10)
y = frexp(x)
y
# this is now true by construction
x==y[,1] *2^y[,2]
```

`HTML.sensitivity.summary`*HTML methods for perturbation analysis*

Description

HTML pretty printing methods for use with R2HTML package

Usage

```
HTML.sensitivity.summary(x,quiet=TRUE,...)
HTML.sensitivity.sim.summary(x,...)
HTML.sensitivity.anova(x,...)
```

Arguments

<code>x</code>	a sensitivity summary object
<code>quiet</code>	show debugging output
<code>...</code>	arguments to pass to default functions

Value

None

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

See Also as `perturb`, `HTML`

Examples

```
## Not run:
if(require(R2HTML,quietly=TRUE)){
  sp=summary(perturb(longley,lm,Employed~.))
  HTML(sp,file="") # HTML Output
}

## End(Not run)
```

modelsCompare
Functions for comparing results: LRE, compare, agrees

Description

Use these functions to compare a set of results from an analysis, to check if they agree to a certain number of significant digits.

Usage

```
# compare numeric values against correct values
LRE(x,correct, use.LAE=TRUE, cutoff=16)

# compare the results from two possibly identical models
modelsAgree(model, model2=NULL, digits=3, ...)
  modelsCompare(model, model2=NULL, param.function=modelBetas,
  similarity.function=LRE, summary.function=min)

# extract betas or entire coefficient summaries
modelBetas(model)
modelSummary(model)
```

Arguments

<code>x</code>	vector of computed values
<code>correct</code>	vector of correct values
<code>use.LAE</code>	use log absolute error if <code>\$c.i\$==0</code> . If false, returns NA where <code>\$c.i\$==0</code>
<code>cutoff</code>	In LRE results, convert INF to cutoff. Since perfect agreement yields an infinite LRE otherwise. Default value of 16 is based on the number of significant digits for doubles.
<code>model</code>	first model for comparison, or if <code>model2=NULL</code> , supply a list of models for comparison
<code>model2</code>	second model for comparison, if
<code>digits</code>	number of digits to use for comparison
<code>param.function</code>	function used to extract model parameters for comparison
<code>similarity.function</code>	function used to compute similarity between sets of model parameters
<code>summary.function</code>	function used to summarize differences
<code>...</code>	parameters to pass from <code>modelsAgree</code> to <code>modelsCompare</code>

Details

`modelsAgree` is a convenience function that calls `modelsCompare` to summarize similarities between models results.

Value

Returns a new vector of log-relative-errors (log absolute error where $\log_{10}(\text{error})$). The resulting values are roughly interpretable as the number of significant digits of agreement between c and x . Larger numbers indicate that x is a relatively more accurate approximation of c . Numbers less than or equal to zero indicate complete disagreement between x and c .

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/, Michael McDonald

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

See Also as `ttst`

Examples

```
# simple LRE examples
LRE(1.001,1) # roughly 3 significant digits agreement
LRE(1,1) # complete agreement
LRE(20,1) # complete disagreement

# compare two models

if (is.R()) {
  hasNmle=require(nlme,quietly=TRUE)
} else {
  hasNmle=require(nlme3,quietly=TRUE)
}

if (hasNmle) {
  O2<-Orthodont
  O3<-Orthodont

  O2[,2]<-O2[,2]+.1
  O3[,2]<-O2[,2]+.11
  if (is.R()) {
    fm1<- lmList(Orthodont)
    fm2<- lmList(O2)
    fm3<- lmList(O3)
  } else {
    # workaround bug in lmList in S-Plus
    fm1<- eval(substitute(lmList(Orthodont)))
    fm2<- eval(substitute(lmList(O2)))
    fm3<- eval(substitute(lmList(O3)))
  }
}
```

```

}

# do the three models agree?
print(modelsAgree(list(fm1, fm2, fm2)))

# show details of disagreement between first 2 models
print(modelsCompare(fm1, fm2))

#compare betas at 2 significant digits
print(modelsAgree(fm1, fm2, digits=2, param.function=modelBetas))
#compare betas at 1 significant digit
print(modelsAgree(fm1, fm2, digits=1, param.function=modelBetas))
}

```

plot.sensitivity *Generic methods for perturbations.*

Description

Perturbations objects can be printed, summarized, and plotted .

Usage

```

plot.sensitivity(x, ask=dev.interactive(), ...)
anova.sensitivity(object, ...)
print.sensitivity(x, quiet=TRUE, ...)
summary.sensitivity(object, ...)

```

Arguments

<code>x</code>	perturb object to plotted or printed
<code>object</code>	perturb object to be summarized, or presented as an anova
<code>...</code>	additional parameters passed to xxx.default functions
<code>ask</code>	whether to pause and ask between plots
<code>quiet</code>	whether to print low-level details of perturb object

Value

see the related generic functions

Note

A perturb object can be summarized, printed, plotted, or summarized via anova if applicable. The idiom `plot(anova(perturb(...)))` will plot the anova summaries, as well. And the `plot(summary(perturb(...)))` will plot the sensitivity of the coefficients. See `perturb` for an example.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

perturb

Examples

```
# see perturb()
```

psim *perform zelig simulations for perturbed models*

Description

This performs `sim()` for perturbed models.

Usage

```
psim(object,x=setx(object),...)  
setx.sensitivity(obj,...)
```

Arguments

object	an object of type <code>perturb</code> , as returned from <code>pzelig()</code>
obj	an object of type <code>perturb</code> , as returned from <code>pzelig()</code>
x	an object returned from <code>setx()</code>
...	arguments to pass to default functions

Value

PSim returns an object containing predictive simulations for each perturbation. Setx returns a setx object. See `sim` and `setx` in the Zelig package.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

See Also as `perturb`, `PTBdefault`, `setx`, `sim`, `pzelig`

Examples

```
# see the example in pzelig()
```

PTBdefault

Returns a default perturbation function for a given vector.

Description

This function returns a function that can be used to repeatedly perturb the vector given to it.

Usage

```
PTBdefaultfn(vec, q = 0.99)  
PTBdefault(vec, q=0.99)
```

Arguments

`vec` the vector which will be subject to perturbation
`q` for discrete vectors, the reclassification probability. For continuous vectors, perturbations will have 1-q relative uniform noise added.

Details

For numeric discrete values, and ordered factors, `reclass.mat.diag` will be used to generate the default reclassification matrix. For character vectors, unordered factors, and logicals, `relass.mat.random` is used.

Value

A function that can be used to perturb the vector.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

perturb, PTBdiscrete, PTBus, reclass.mat.random

Examples

```
x=1:100
# perturb using the default method
rpx=replicate(100,PTBdefault(x),simplify=FALSE)
# how many matches to original vector? mean should be close to 95
if (is.R()) {
  matches <-sapply(rpx,function(y)(sum(y==x))) # how many matches to original vector
} else {
  # Splus variation
  matches <-sapply(rpx,substitute(function(y)(sum(y==x))))
}
summary(matches)
# This produces equivalent results, but is faster,
# since reclass matrices are not recalculated on each replication

fx=PTBdefaultfn(x,q=.95)
rpx=replicate(100,fx(x),simplify=FALSE)
if (is.R()) {
  matches <-sapply(rpx,function(y)(sum(y==x))) # how many matches to original vector
} else {
  # Splus variation
  matches <-sapply(rpx,substitute(function(y)(sum(y==x))))
}
summary(matches)
```

PTBdiscrete	<i>Function to perturb vectors of discrete numeric values or factors, logicals, characters</i>
-------------	--

Description

This uses a reclassification matrix to perturb a vector discrete values.

Usage

```
PTBdiscrete(x, r=NULL, q=0.99) # for factors, discrete numeric
```

Arguments

x	the vector to be perturbed
r	a reclassification matrix produced by <code>reclass.mat.diag</code> or <code>reclass.mat.random</code> or <code>reclassify</code> (from the separate perturb library)
q	q-value for reclassification, if r is NULL

Details

This perturbs discrete vectors by reclassifying them at random based on a cumulative probability reclassification matrix.

By default, if `r` is not supplied a default matrix will be supplied that has probability=`q` of keeping the same values, using the same rules as `PTBdefault`.

As an alternative to classification for numeric vectors, see `PTBu` and related functions for methods that add random noise to a vector.

Value

Returns a new vector `x'` with elements randomly reclassified according to `r`.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

`perturb`, `PTBn`, `PTBdefault`, `reclass.mat.random`, `reclass.mat.diag`

Examples

```
x=ceiling(runif(1:100)*3) # vector taking on 3 discrete levels
rx = reclass.mat.random(3,.95) # reclassification matrix, prob of change = .05
rpx=replicate(100,PTBdiscrete(x,rx),simplify=FALSE) # 100 perturbations
if (is.R()) {
  matches <-sapply(rpx,function(y)(sum(y==x))) # how many matches to original vector
} else {
  # Splus variation
  matches <-sapply(rpx,substitute(function(y)(sum(y==x))))
}
summary(matches) # mean should be close to .95
```

 PTBi

functions to add random noise to a vector

Description

These functions add noise to a vector. These are helper functions for `perturb`. They are used to select the type and magnitude of noise applied to each vector in the data frame, when running the perturbation sensitivity analysis. Use them only if they are substantively justified – you can also supply custom functions for use with this perturbation framework.

Usage

```

#
# identity function
#

PTBi(x,size=1) # identify function, returns original vector
# size is a dummy value, it is ignored
#
# minimal noise
#

PTBms(x, size=1) # adds minimal, scaled, noise
PTBmsb(x, size=1, lbound=0, ubound=1) # adds minimal, scaled, noise,
# resulting vector truncated to bounds
PTBmsbr(x, size=1, lbound=0, ubound=1) # adds minimal, scaled noise, out of-bounds elements
# of vector resamples until within bounds

#
# Normally distributed noise
#

PTBn(x, size = 1) # adds normally distributed noise
PTBnc(x, size = 1) # normally distributed noise,
# recentered to guarantee mean-zero disturbance
PTBns(x, size = 1) # adds normally distributed, scaled, noise

PTBnbr(x, size = 1, lbound=0, ubound=1)#
PTBnsbr(x, size = 1, lbound=0, ubound=1)# adds normally (scaled) distributed noise
# out of-bounds elements
# of vector resampled until within bounds

PTBnbrr(x, size = 1, lbound=0, ubound=1)#
PTBnsbrr(x, size = 1, lbound=0, ubound=1) # adds normally distributed,scaled noise
# noise rescaled to bounds, and then
# out of bounds elements
# resampled until within bounds

#
# Uniformly distributed noise

PTBu(x, size = 1) # adds uniformly distributed noise
PTBuc(x, size = 1) # uniformly distributed noise,
# recentered to guarantee mean-zero disturbance
PTBus(x, size = 1) # adds uniformly distributed, scaled, noise

PTBubr(x, size = 1, lbound=0, ubound=1)#
PTBusbr(x, size = 1, lbound=0, ubound=1)# adds uniformly, scaled, distributed noise

```

```

# out of-bounds elements
# of vector resampled until within bounds

PTBubrr(x, size = 1, lbound=0, ubound=1)#
PTBusbrr(x, size = 1, lbound=0, ubound=1) # adds uniformly distributed, scaled noise
# noise rescaled to bounds, and then
# out of bounds elements
# resampled until within bounds

```

Arguments

<code>x</code>	vector to be perturbed
<code>size</code>	magnitude of noise
<code>ubound</code>	upper bound, can be a scalar or a vector of the same length as <code>x</code>
<code>lbound</code>	lower bound, can be a scalar or a vector of the same length as <code>x</code>

Details

Noise is randomly adds to each observation in the vector according to the chosen distribution function: - 'minimal' adds `.Machine$double.ep`) or subtracts `.Machine$double.neg.eps` - 'uniform' adds uniformly distributed random `[-size/2,+size/2]` - 'normal' adds `Normal(mean=0,stddev=size)`

Noise is asymptotically mean zero. To guarantee mean-zero noise for a particular sample, use the "centered" helper functions, which adjust the noise vector to be approximately mean zero.

The 'scaled' functions, multiply each random deviate by `x_i` for each observation in `x`. This is useful if measurement error is heteroscedastic.

Some helper functions can return a noisy vector guaranteed to be within given upper and lower bounds. This can be done through truncation, through resampling out-of-bounds observations, or by reducing the amount of noise added to entries close to the bounds.

Value

Returns a new vector `x'`, with the same length as `x`.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

perturb

Examples

```
x=1:1000
x.i = PTBi(x)
x.m = PTBms(x)
x.u = PTBu(x,size=.01)
x.us = PTBus(x,size=.01)
x.uc = PTBuc(x,size=.01)

sum(x-x.i) # should be 0, identity transform
sum(which(x!=x.i)) # also 0

sum(x-x.m) # should be quite small, very close to 0
length(which(x!=x.m)) # should be near 1000

sum(x-x.u) # should be small
length(which(x!=x.m)) # should be near 1000

sum(x-x.us) # should be small
length(which(x!=x.m)) # should be near 1000

sum(x-x.uc) # should be near 0
length(which(x!=x.m)) # should be near 1000
```

PTBmu.gen

generator functions for multiple rounds of noise

Description

These functions generate a function that will apply multiple rounds of noise. These are helper functions for `perturb`, and generate functions of the form of `PTBn`. They are used to select the type and magnitude of noise applied to each vector in the data frame, when running the perturbation sensitivity analysis. Use them only if they are substantively justified – you can also supply custom functions for use with this perturbation framework.

Usage

```
PTBmu.gen(reps=1) # return a perturbation function that
PTBmn.gen(reps=1) # applies multiple rounds of noise
```

Arguments

`reps` number of rounds of noise to apply

Details

Used to generate functions for use in `perturb`

Value

Returns a function of the form `PTBn` or `PTBi`, with the modification that *reps* rounds of noise are applied for each perturbation.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

`perturb`, `PTBi`

Examples

```
x=1:1000
f1=PTBmu.gen(); # should be roughly equivalent to PTBu()
x.u = f1(x,size=1)
mean(x-x.u) #should be small
f2=PTBmu.gen(reps=100); # multiple disturbances tend to cancel eachother out
x.u2 = f2(x,size=1)
mean(x-x.u2) #should be smaller
```

`reclass.mat.diag` *Function to produce reclassification matrices*

Description

This returns a cumulative probability matrix useful for reclassifying discrete variables with `PTBdiscrete`.

Usage

```
reclass.mat.diag(n, q)
```

Arguments

`n` number of levels of factor, character vector, or discrete values
`q` probability of reclassifying to same state

Details

For reclass.mat.diag, the transition probability of changing from level i to level j is q if $i=j$, 0 if $\text{abs}(i-j)>1$ and q , $1-q/2$ if $\text{abs}(i-j)>1$ and $1>i>n$ and $1-q$ otherwise

For reclass.mat.random, the transition probability of changing from level i to level j is q if $i=j$, $(1-q)/(n-1)$ otherwise.

Value

returns a matrix of cumulative probability distributions for reclassifying each level to another level

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

perturb, reclass.mat.random

Examples

```
x<-ceiling(runif(1:100)*3) # vector taking on 3 discrete levels
rx <-reclass.mat.random(3,.95) # reclassification matrix, prob of change = .05
rpx<-replicate(10,PTBdiscrete(x,rx),simplify=FALSE) # 100 perturbations
if (is.R()) {
  matches <-sapply(rpx,function(y)(sum(y==x))) # how many matches to original vector
} else {
  # Splus variation
  matches <-sapply(rpx,substitute(function(y)(sum(y==x))))
}
summary(matches) # mean should be close to .95
```

runifS

Function to return TRUE (not pseudo) random numbers, based on system and networked entropy collection.

Description

This function makes use of hardware-generated entropy to supply true random numbers. Entropy collection is slow, so its primary use to provide random numbers for cryptographic key generation and for setting the seed value for subsequent PRNG generation.

Usage

```
runifT(n,min=0,max=1,maxTries=10,silent=TRUE)
runifS(n,...,period=10000,maxTries=10,silent=TRUE)
resetSeed(maxTries=10,silent=TRUE)
```

Arguments

n	number of observations. If 'length(n) > 1', the length is taken to be the number required.
min,max	lower and upper limits of the distribution
...	pass on to runif
period	number of random numbers to generate before reseeding
maxTries	number of times to attempt to contact entropy source before giving up
silent	whether to report failures of entropy source generators

Details

runifS returns pseudo-random numbers, reseeding with a true random number every period draws. **resetSeed** resets R's own PRNG seed, **set.seed**, supplying a true random number **runifT** returns true random numbers in a uniform distribution. This can easily exhaust entropy sources, so for larger samples, **runifS** is usually preferred.

These routines provides true random numbers by accessing external entropy collectors. Currently entropy can be retrieved from two different sources. (1) The "Hotbits" server <http://www.fourmilab.ch/hotbits/> supplies random bytes based on radioactive decay. (2) The bit server on random.org, and (3) On Unix systems, the the kernel gathers environmental noise from device drivers and other sources into an entropy pool, which can be accessed through '/dev/random'.

Entropy is retrieved in blocks from the sources (each source having a different preferred block size). If the sources do not return a block within the R timeout value (see **options("timeout")**) entropy requests will be repeated up to **maxTries** times per source. If no entropy is available, pseudo random numbers will be returned using **runif()** (or a seed will be set based on **Sys.time**, for **resetSeed**, and a warning issued.

Value

runifS, **runifT** Returns a vector of uniformly distributed true random numbers. **resetSeed** returns the status of **set.seed()**

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

See Also `runif`, `options`, `.Random.seed`,

Examples

```
# note, if you are using Windows, you must be on-line
# to get to entropy generator, will fall-back to pseudo-random
# numbers when off-line

# reset the seed for runif() based on a true random value

resetSeed()
y=runif(100000)
ty= table(trunc(5*y))
print(ty)
if (is.R()) {
  chisq.test(ty)
} else {
  chisq.test(t(cbind(as.numeric(names(ty)),as.matrix(ty))))
}

# generate true random values directly (may block for long periods if
# if entropy pool is empty)
y=runifS(100000)
ty= table(trunc(5*y))
print(ty)
if (is.R()) {
  chisq.test(ty)
} else {
  chisq.test(t(cbind(as.numeric(names(ty)),as.matrix(ty))))
}

## Not run:
y=runifT(100)
ty= table(trunc(5*y))
print(ty);
chisq.test(ty)

## End(Not run)
```

Description

Perform the Schnabel-Eskow Choleksy Matrix Decomposition

Usage

```
sechol(A, tol=.Machine$double.eps, silent=TRUE)
```

Arguments

<code>A</code>	a square matrix object which can be non-singular.
<code>tol</code>	a tolerance parameter
<code>silent</code>	print debugging messages

Details

The algorithm of Schnabel and Eskow (Schnabel, R. B. and Eskow, E. 1990. "A New Modified Cholesky Factorization." *SIAM Journal of Scientific Statistical Computing* 11, 1136-58.) improves the Gill/Murray approach of incrementing diagonal values of a singular matrix sufficiently that Cholesky steps can be performed. The algorithm is based on applying the Gerschgorin Circle Theorem to reduce the infinity norm of the incrementing matrix. The strategy is to calculate delta values that reduce the *overall* difference between the singular matrix and the incremented matrix.

Value

A Cholesky decomposition of the matrix C+E where C is the original non-singular matrix and E is the minimal diagonal increment according to the Gerschgorin Circle Theorem. If the input matrix is non-singular, then the Cholesky decomposition of C is returned.

Author(s)

Jeff Gill

References

Schnabel, R. B. and Eskow, E. 1990. "A New Modified Cholesky Factorization." *SIAM Journal of Scientific Statistical Computing* 11, 1136-58.

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

ginv solve chol svd qr

Examples

```
# compare with chol() on a non-singular matrix
S <- matrix(c(2,0,2.4,0,2,0,2.4,0,3),ncol=3)
chol(S)
TS<- sechol(S)
TS
t(TS)
```

```

# an example with a singular matrix
S <- matrix(c(2,0,2.5,0,2,0,2.5,0,3),ncol=3)
TS<-sechol(S)
TS
t(TS)

# another example with a singular matrix
S <- matrix(c(2,0,10,0,2,0,10,0,3),ncol=3)
TS <- sechol(S)
TS
t(TS)

```

sensitivityZelig *Perturbations-based Sensitivity Analysis of Zelig Models*

Description

This is a wrapper to perturb analyses launches through Zelig, with extension functions to summarize results.

Usage

```

sensitivityZelig(z, ptb.R=50, ptb.ran.gen=NULL, ptb.s=NULL, explanatoryOnly=FALSE, summarize=FALSE,
pzelig(...))

```

Arguments

<code>z</code>	an analysis output by <code>zelig</code>
<code>ptb.R</code>	number of replications for perturbation analysis
<code>ptb.ran.gen</code>	a single function, or a vector of functions to be used to perturb each vector see PTBi
<code>ptb.s</code>	a size, or vector of sizes, to be used in the vector perturbation functions
<code>explanatoryOnly</code>	a flag indicating whether explanatory variables should be perturbed by default
<code>summarize</code>	if true, return a sensitivity summary, as would <code>summary(sensitivityZelig())</code> .
<code>simulate</code>	if true, return a sensitivity summary, and a zelig simulation, as would <code>psim(sensitivityZelig())</code> . This decreases system memory use, and can significantly speed up analysis for large datasets.
<code>simArgs</code>	a list of arguments to pass to <code>psim</code> , if <code>simulate</code> is true
<code>...</code>	args as in <code>sensitivityZelig</code>
<code>ptb.rangen.ismatrix</code>	If true, expects <code>ptb.ran.gen</code> to be a matrix function, for use with correlated noise structure. See below

Details

Uses `sensitivity` to perform sensitivity analysis on a `zelig` model.

Value

Returns a list which contains the result of each model run. Along with attributes about the settings used in the perturbations. Use `summary` to summarize the results or extract a matrix of of the model parameters across the entire set of runs.

Note

This was originally called "pzelig". If `ptb.ran.gen` is not specified, then `PTBdefault` will be used, with `q=ptb.s`, for each explanatory variable in the input data. By default, response variables will also be perturbed, unless `explanatoryOnly` is true.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmhc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmhc.harvard.edu/numerical_issues/

See Also

See Also as `perturb`, `PTBdefault`, `zelig`, `psim`

Examples

```
# Sensitivity analysis of the classic longley data, as run through zelig.
#
# Note: Compare to the example documented under the main perturb page.
# example requires Zelig.

if(!is.R()){
  # splus version of require ignores quietly!
  ow<-options(warn=-1)
}
if (require(Zelig,quietly=TRUE)) {
  if (is.R()) {
    data(longley)
  } else {
    # accuracy supplies longley for SPLUS automagically, for examples only

    options(ow)
  }
}
```

```

zelig.out=zelig(Employed~., "ls", longley) # run longley regression
# default settings
perturb.zelig.out = sensitivityZelig(zelig.out)

# Note: without zelig, the preceding would be: perturb.out = perturb(longley, lm, Employed~.)

# plots the perturbed lm replications individually
plot(perturb.zelig.out)
# summarizes the overall sensitivity of coefficient estimates
summary(perturb.zelig.out)
plot(summary(perturb.zelig.out))

# since the "ls" model relies on "lm" which supports anova, can summarize
# using anova
anova(perturb.zelig.out)
plot(anova(perturb.zelig.out))

# Up to this point, we could have done this without Zelig... here s the fun part.

# set values of explanatory variables
setx.out=setx(perturb.zelig.out, Year=1955)

# Note: could also have used setx.out=setx(z, Year=1955) instead of the line above,
#       or simply called psim() below without setx.out, which would default to using setx(perturb.zel

# use simulation to predict value of explanatory variable based on
#
sim.perturb.zelig.out = psim(perturb.zelig.out, setx.out)

# this plots the profile of the predicted distribution of the explanatory variable, Employed
# around the point wher Year=1955, and other explanatory variables are at their midpoints

plot(sim.perturb.zelig.out)

# this provides a summary of the predicted distribution of the explanatory variable

print(summary(sim.perturb.zelig.out))

}
if (!is.R()) {
  options(ow)
}

```

Description

This function replicates an statistical analysis using slightly perturbed versions of the original input data, and then analyzes the sensitivity of the model to such changes. This can be

used to draw attention to inferences that cannot be supported confidently given the current data, model, and algorithm/implementation,

Usage

```
sensitivity(data, statistic, ..., ptb.R = 50, ptb.ran.gen = NULL, ptb.s = NULL,
summarize=FALSE, keepData = FALSE, ptb.rangen.ismatrix=FALSE)
perturb(...)
```

Arguments

<code>data</code>	data matrix to be perturbed
<code>statistic</code>	statistic model to be run on data
<code>...</code>	additional arguments to statistical model
<code>ptb.R</code>	number of replications for perturbation analysis
<code>ptb.ran.gen</code>	a single function, or a vector of functions to be used to perturb each vector see PTBi
<code>ptb.s</code>	a size, or vector of sizes, to be used in the vector perturbation functions
<code>summarize</code>	if true, return a sensitivity summary, as would <code>summary(sensitivity())</code> . This reduces system memory use and can significantly speed up the analysis of large datasets.
<code>keepData</code>	for debugging, store data for each perturbation
<code>ptb.rangen.ismatrix</code>	If true, expects <code>ptb.ran.gen</code> to be a matrix function, for use with correlated noise structure. See below

Details

Sensitivity to numerical inaccuracy, and measurement error is very hard to measure formally. This empirical sensitivity tests draws attention to inferences that cannot be supported confidently given the current data, model, and algorithm/implementation,

The empirical approach works by replicating the original analysis while slightly perturbing the original input data in different ways. The sensitivity of the model estimates (e.g. estimated coefficients, standard errors and log-likelihoods) are then summarized.

The sensitivity analysis cannot be used to prove the accuracy of a particular method, but is useful in drawing attention to potential problems. Further experimentation and analysis may be necessary to determine the specific cause of the problem: numerical instability, statistical sensitivity to measurement error, or ill-conditioning.

Value

Returns a list which contains the result of each model run. Along with attributes about the settings used in the perturbations. Use `summary` to summarize the results or extract a matrix of of the model parameters across the entire set of runs.

Note

If `ptb.ran.gen` is not specified, then `PTBdefault` will be used, with `q=ptb.s`, for each variable in the input data.

Note "sensitivity" was originally called "perturb". The name was changed to avoid a conflict with another module, introduced afterwards.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

See Also

See Also as `PTBi`, `sensitivityZelig`, `PTBdefault`, `PTBdiscrete`

Examples

```
# Examine the sensitivity of the GLM from Venables & Ripley (2002, p.189)
# as described in the glm module.
#
# Perturb the two independent variables using +/- 0.025%
# (relative to the size of each observations)
# uniformly distributed noise. Dependent variable is not being modified.
#
# Summary should show that estimated coefficients are not substantively affected by noise.

if (!is.R()){
  # workaround MASS data bug in SPLUS
  require(MASS,quietly=TRUE)
}

data(anorexia,package="MASS")
panorexia = sensitivity(anorexia, glm, Postwt ~ Prewt + Treat + offset(Prewt),
  family=gaussian,
  ptb.R=100, ptb.ran.gen=list(PTBi,PTBus,PTBus), ptb.s=c(1,.005,.005) )
summary(panorexia)

# Use classic longley dataset. The model is numerically unstable,
# and much more sensitive to noise. Smaller amounts of noise tremendously
# alter some of the estimated coefficients:
#
# In this example we are not perturbing the dependent variable (employed) or
# the year variable. So we assign then PTBi or NULL in ptb.ran.gen )
```

```

data(longley)
plongley = sensitivity(longley,lm,Employed~.) # defaults

# Alternatively, choose specific perturbation functions
#
plongley2 = sensitivity(longley,lm,Employed~., ptb.R=100,
  ptb.ran.gen=c(list(PTBi), replicate(5,PTBus,simplify=FALSE),list(PTBi)), ptb.s=c(1,replicate(5,.001)),

# summarizes range
sp=summary(plongley)
print(sp)
plot(sp) # plots boxplots of the distribution of the coefficients under perturbatione

# models with anova methods can also be summarized this way
anova(plongley)

## Not run:
# plots different replications
plot(plongley) # plots the perturbed replications individually, pausing in between

# plots anova results (where applicable)
plot(anova(plongley))

## End(Not run)

# look in summary object to extract more ...
names(attributes(sp))

# print matrix of coefficients from all runs
coef= attr(sp,"coef.betas.m")
summary(coef)

# Example where model does not accept a dataset as an argument...

# MLE does not accept a dataset as an argument, so need to
# create a wrapper function
#
#
if (is.R()) {
  library(stats4)
  mleD<-function(data,lld,...) {
    # construct LL function with embedded data
    f=formals(lld)
    f[1]=NULL
    ll <-function() {
      cl=as.list(match.call())
      cl[1]=NULL
      cl$data=as.name("data")
      do.call(lld,cl)
    }
  }
}

```

```

    }
    formals(l1)=f

    # call mle
    mle(l1,...)
  }

  dat=as.data.frame(cbind( 0:10 , c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8) ))

  llD<-function(data, ymax=15, xhalf=6)
    -sum(stats::dpois(data[[2]], lambda=ymax/(1+data[[1]]/xhalf), log=TRUE))

  print(summary(sensitivity(dat, mleD,llD)))
}

# An example of using correlated noise by supplying a matrix noise function
# Note that the function can be an anonymous function supplied in the call itself

if (require(MASS,quietly=TRUE)) {

  plongleym=sensitivity(longley,lm,Employed~.,
    ptb.rangen.ismatrix=TRUE,
    ptb.ran.gen=
      function(x,size=1){
        mvrnorm(n=dim(x)[1],mu=rep(0,dim(x)[1]),
          Sigma=matrix(.9,nrow=dim(x)[1],ncol=dim(x)[1])*size+x}
      )
    print(summary(plongleym))
  }
}

```

 starr

starr global optimum test

Description

Implements the Starr test for identification of the global optimum of a likelihood surface.

Usage

```
starr(betas, tol=.0001, dmethod="euclidean")
```

Arguments

<code>betas</code>	Vector of parameter values
<code>tol</code>	Tolerance distance between two parameter vectors to consider as "unique" optimum
<code>dmethod</code>	method used to compute distance between two parameter vectors

Details

`\textttstarr` computes the probability that a local optimum, which may or may not be the global optimum, of a likelihood function has not been observed. The probability is generated by counting the observed number of "basins of attraction" - starting values that lead to an local optimum.

Value

`\textttstarr` is a class "double" with value equal to the probability than a local optimum is unobserved.

Note

`\textttstarr` is given a vector of `\textttbetas` parameter values identifying local optima resultant of a search algorithm, such as `\textttmle` or `\textttnl`. The starting values used to generate `\textttbetas` can be a grid, or for larger n-dimensional parameter spaces, may be randomly chosen. The parameter values identifying a local optimum are passed to `\textttstarr` for each set of starting values that lead to that local optimum.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/ , Michael McDonald

References

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons. http://www.hmdc.harvard.edu/numerical_issues/

Finch, S. J., Mendell, N. R., Thode, H. C., Jr. 1989. "Probabilistic Measures of Adequacy of a Numerical Search for a Global Maximum." *Journal of the American Statistical Association* **84**, 1020-3.

Starr, N. 1979. "Linear Estimation of the Probability of Discovering a New Species." *Annals of Statistics* **7**, 644-52.

Examples

```
x=rbind(c(1,1,1), c(1,2,1), c(1,1.1,1), c(1,2,1), c(3,4,5));
starr(rbind(1,1,2,2));
```

```
#BODmodel
```

```

BOD <-
structure(list(Time = c(1, 2, 3, 4, 5, 7), demand = c(8.3, 10.3,
19, 16, 15.6, 19.8)), .Names = c("Time", "demand"), row.names = c("1",
"2", "3", "4", "5", "6"), class = "data.frame", reference = "A1.4, p. 270")

stval=expand.grid(A = seq(10, 100, 10), lrc = seq(.5, .8, .1))
llfun<-function(A,lrc,BOD)
  -sum((BOD$demand - A*(1-exp(-exp(lrc)*BOD$Time)))^2)
lls=NULL
for (i in 1:nrow(stval)) {
  lls = rbind(lls, llfun(stval[i,1], stval[i,2],BOD))
}
fm1 <- nls(demand ~ A*(1-exp(-exp(lrc)*Time)),
  data = BOD, start = c(A = 20, lrc = log(.35)))
ss = -sum(resid(fm1)^2)
dehaan(lls, ss)

llb=NULL
for (i in 1:nrow(stval)) {
llb = rbind(llb,coef(
  nls(demand ~ A*(1-exp(-exp(lrc)*Time)),
    data = BOD, start = c(A=stval[i,1], lrc = stval[i,2])))
}
starr(llb)

```

Index

- *Topic **array**
 - sechol, 21
- *Topic **datagen**
 - runifS, 19
- *Topic **debugging**
 - Distribution Test Data, and log relative error function, 3
 - modelsCompare, 8
- *Topic **distribution**
 - plot.sensitivity, 10
 - PTBdefault, 12
 - PTBdiscrete, 13
 - PTBi, 14
 - PTBmu.gen, 17
 - reclass.mat.diag, 18
 - runifS, 19
- *Topic **htest**
 - dehaan, 2
 - starr, 29
- *Topic **manip**
 - dehaan, 2
 - frexp, 5
 - starr, 29
- *Topic **math**
 - dehaan, 2
 - frexp, 5
 - starr, 29
- *Topic **misc**
 - dehaan, 2
 - Distribution Test Data, and log relative error function, 3
 - frexp, 5
 - HTML.sensitivity.summary, 6
 - modelsCompare, 8
 - plot.sensitivity, 10
 - psim, 11
 - PTBdefault, 12
 - PTBdiscrete, 13
 - PTBi, 14
 - PTBmu.gen, 17
 - reclass.mat.diag, 18
 - runifS, 19
 - sensitivityZelig, 23
 - sensitivty, 25
 - starr, 29
- *Topic **optimize**
 - HTML.sensitivity.summary, 6
 - plot.sensitivity, 10
 - psim, 11
 - PTBdefault, 12
 - PTBdiscrete, 13
 - PTBi, 14
 - PTBmu.gen, 17
 - reclass.mat.diag, 18
 - sechol, 21
 - sensitivityZelig, 23
 - sensitivty, 25
 - .Random.seed, 21
- anova.sensitivity (*plot.sensitivity*), 10
- chisqst (*Distribution Test Data, and log relative error function*), 3
- chol, 22
- dehaan, 2
- Distribution Test Data, and log relative error function, 3
- frexp, 5
- ftst (*Distribution Test Data, and log relative error function*), 3
- gammatst (*Distribution Test Data, and log relative error function*), 3
- ginv, 22

- HTML, 7
- HTML.sensitivity.anova
(*HTML.sensitivity.summary*), 6
- HTML.sensitivity.sim.summary
(*HTML.sensitivity.summary*), 6
- HTML.sensitivity.summary, 6
- LRE (*modelsCompare*), 8
- modelBetas (*modelsCompare*), 8
- modelsAgree (*modelsCompare*), 8
- modelsCompare, 8
- modelSummary (*modelsCompare*), 8
- normtst (*Distribution Test Data, and
log relative error function*),
3
- options, 21
- perturb, 7, 11–14, 17–19, 24
- perturb (*sensitivity*), 25
- plot.sensitivity, 10
- print.sensitivity (*plot.sensitivity*),
10
- psim, 11, 24
- PTBdefault, 12, 12, 14, 24, 27
- PTBdefaultfn (*PTBdefault*), 12
- PTBdiscrete, 13, 13, 27
- PTBi, 14, 18, 23, 26, 27
- PTBmn.gen (*PTBmu.gen*), 17
- PTBms (*PTBi*), 14
- PTBmsb (*PTBi*), 14
- PTBmsbr (*PTBi*), 14
- PTBmu.gen, 17
- PTBn, 14, 17, 18
- PTBn (*PTBi*), 14
- PTBnbr (*PTBi*), 14
- PTBnbr (*PTBi*), 14
- PTBnc (*PTBi*), 14
- PTBns (*PTBi*), 14
- PTBnsbr (*PTBi*), 14
- PTBnsbr (*PTBi*), 14
- PTBu (*PTBi*), 14
- PTBubr (*PTBi*), 14
- PTBubr (*PTBi*), 14
- PTBuc (*PTBi*), 14
- PTBus, 13
- PTBus (*PTBi*), 14
- PTBusbr (*PTBi*), 14
- PTBusbr (*PTBi*), 14
- pzelig, 12
- pzelig (*sensitivityZelig*), 23
- qr, 22
- reclass.mat.diag, 14, 18
- reclass.mat.random, 13, 14, 19
- reclass.mat.random
(*reclass.mat.diag*), 18
- resetSeed (*runifS*), 19
- runif, 21
- runifS, 19
- runifT (*runifS*), 19
- sechol, 21
- sensitivity (*sensitivity*), 25
- sensitivityZelig, 23, 27
- sensitivity, 25
- setx, 12
- setx.sensitivity (*psim*), 11
- sim, 12
- solve, 22
- starr, 29
- summary.sensitivity
(*plot.sensitivity*), 10
- svd, 22
- trueRandom (*runifS*), 19
- ttst, 9
- ttst (*Distribution Test Data, and
log relative error function*),
3
- zelig, 24