Package ‘revealedPrefs’

September 5, 2019

Type Package
Title Revealed Preferences and Microeconomic Rationality
Version 0.4.1
Date 2019-09-03
Author Julien Boelaert <jubo.stats@gmail.com>
Maintainer Julien Boelaert <jubo.stats@gmail.com>
License GPL (>= 3)
Imports pso
Depends Rcpp
LinkingTo Rcpp, RcppArmadillo
NeedsCompilation yes
Repository CRAN
Date/Publication 2019-09-05 02:10:05 UTC

R topics documented:

revealedPrefs-package ................................................................. 2
checkGarp ................................................................. 3
cpLower ................................................................. 5
cpUpper ................................................................. 6
direct_prefs ................................................................. 8
noGarp ................................................................. 9
simGarp ................................................................. 10
simPrefs ................................................................. 11

Index 14
Description


Details

This package is meant for the analysis of (quantity, price) data, eg. of bundles of goods and corresponding prices. It features fast algorithms that make the analysis of large datasets feasible.

Functions `directPrefs` and `indirectPrefs` compute revealed preferences.

Functions `checkWarp`, `checkSarp`, `checkGarp` perform fast non-parametric tests of rationality using the corresponding rationality axioms.

Functions `simWarp`, `simSarp`, `simGarp` and `simPrefs` generate simulated data consistent with a rationality axiom or with a given preference matrix.

Functions `cpLower` and `cpUpper` generate Crawford-Pendakur type bounds on the number of subpopulations and provide the corresponding clusterings.

Author(s)

Julien Boelaert <jubo.stats@gmail.com>

Maintainer: Julien Boelaert <jubo.stats@gmail.com>

References


See Also

See `directPrefs` for computation of preferences, `checkGarp` for rationality tests, `simGarp` for data generation, and `cpUpper` for clustering of data into non-violating subsets.
checkGarp

Examples

# Compute preferences and check rationality on a GARP-violating dataset:
data(noGarp)
indirectPrefs(noGarp$x, noGarp$p)
checkGarp(noGarp$x, noGarp$p)

# Cluster dataset into GARP-consistent subpopulations:
cpUpper(noGarp$x, noGarp$p)

checkGarp

Non-parametric tests of rationality axioms (WARP, SARP, GARP).

Description

Functions for non-parametric tests of compliance with rationality axioms.

Functions checkWarp, checkSarp and checkGarp perform exact tests of the Weak Axiom of Revealed Preferences (WARP), the Strong Axiom of Revealed Preferences (SARP) and the Generalized Axiom of Revealed Preferences (GARP) respectively, to check whether the given data are consistent with utility maximization.

Usage

checkWarp(x, p, afriat.par= 1)
checkSarp(x, p, afriat.par= 1, method= c(“deep”, “floyd”))
checkGarp(x, p, afriat.par= 1, method= c(“floyd”))
## S3 method for class 'axiomTest'
print(x, ...)
## S3 method for class 'axiomTest'
summary(object, ...)

Arguments

x data frame or matrix containing the observed quantities, where each row corresponds to an observation and the columns are types of goods, or an object of class axiomTest for use with print,
p data frame or matrix (of same dimensions as x) containing the corresponding prices,
afriat.par the Afriat parameter, real number in [0,1], which allows a certain level of error in the optimization of choices; default is 1, i.e. no optimization error allowed,
method character string: “deep” for depth-first search (default for Sarp), “floyd” for Floyd-Warshall algorithm (can be very slow for large datasets when no violations are present),
object an object of class axiomTest as returned by eg. checkWarp,
... additional arguments passed to the print and summary methods (unused).
Details

Rationality axioms can be summarized as follows:

WARP: if X directly prefered to Y and X is not equal to Y, then Y cannot be directly prefered to X (WARP is a necessary condition for the existence of a single-valued utility-maximizing demand function consistent with the data).

SARP: if X (in)directly prefered to Y and X is not equal to Y, then Y cannot be (in)directly prefered to X (SARP is a necessary and sufficient condition for the existence of a single-valued utility-maximizing demand function consistent with the data).

GARP: if X (in)directly prefered to Y, then Y cannot be strictly directly prefered to X (GARP is a necessary and sufficient condition for the existence of a (possibly multiple-valued) utility-maximizing demand function consistent with the data).

If WARP or GARP are violated, then SARP is also violated.

Testing of WARP is straightforward by pairwise comparison of revealed preferences. GARP is tested using the Floyd-Warshall algorithm to find the transitive closure of the direct preference relationship. SARP can be tested either using the Floyd-Warshall algorithm, or using a depth-first search that systematically explores the preference relationship in search of a violating cycle.

Value

Functions checkWarp, checkSarp and checkGarp return an object of class axiomTest which may contain the following elements:

violation logical value, TRUE if a violation was found,
path last path taken during depth-first search,
path.strict (for deep method) vector of logical values indicating the strictness of direct preference relations in the last path taken,
v violators vector of indices of axiom-violating observations (only the first violators that were found are reported, this is not a complete list),
strict vector of logical values indicating whether the preferences between reported violators (or path) are strict or not,
direct.violation logical value, TRUE if a violation was found in direct preferences,
type the violated or non-violated rationality axiom,
method method used for the non-parametric test,
afr iat.par Afr iat parameter used in the algorithm.

Author(s)

Julien Boelaert <jubo.stats@gmail.com>

References


Examples

# Check axioms for GARP-violating data:
data(noGarp)
summary(checkGarp(noGarp$x, noGarp$p))

# Check axioms for SARP-consistent data:
data(okSarp)
summary(checkSarp(okSarp$x, okSarp$p))

cpLower

Lower bound on the number of GARP-consistent subpopulations.

Description

The cpLower function computes a Crawford-Pendakur type lower bound on the number of GARP-consistent subpopulations by creating a set of pairwise GARP-violating observations.

Usage

cpLower(x, p, times=1, afriat.par=1)
# S3 method for class 'lowerBound'
print(x, ...)
# S3 method for class 'lowerBound'
summary(object, ...)

Arguments

x data frame or matrix containing the observed quantities, where each row corresponds to an observation and the columns are types of goods, or an object of class lowerBound to be used with print,
p data frame or matrix (of same dimensions as x) containing the corresponding prices,
times number of times the algorithm is run (the final result is the best of times results, i.e. highest number of clusters found),
afriat.par the Afriat parameter, a real number in [0,1], which allows a certain level of error in the optimization of choices; default is 1, i.e. no optimization error allowed,
object object of class lowerBound as returned by cpLower,
... additional arguments passed to the print and summary methods (unused).

Details

For each run of the algorithm, a random permutation of the observations is drawn, and one by one each observation is pairwise-tested against all previously found violators. If the current observation is found pairwise-inconsistent with all previously found violators it is added to the set of violators.
Value

cpLower returns an object of class `lowerBound` which contains the following elements:

- `violators` numeric vector containing the indices of observations that are pairwise GARP-inconsistent,
- `n.clust` lower bound on the number of types,
- `hist.n.types` numeric vector containing the history of numbers of types found during multiple runs of the algorithm.
- `n.types` lower bound on the number of types,
- `afriat.par` Afriat parameter used in the algorithm.

Author(s)

Julien Boelaert <jubo.stats@gmail.com>

References


See Also

See `cpUpper` for the upper bound on the number of types.

Examples

```r
# Lower bound for GARP-violating data:
data(noGarp)
cp.low <- cpLower(noGarp$x, noGarp$p)
cp.low
```

Description

The `cpUpper` function computes a Crawford-Pendakur type upper bound on the number of GARP-consistent subpopulations, and performs clustering of the data.

Usage

```r
cpUpper(x, p, times = 1, afriat.par = 1, method = c("fastfloyd", "floyd"))
## S3 method for class 'upperBound'
print(x, ...)  
## S3 method for class 'upperBound'
summary(object, ...)
```
Arguments

x data frame or matrix containing the observed quantities, where each row corresponds to an observation and the columns are types of goods, or an object of class upperBound to be used with print,
p data frame or matrix (of same dimensions as x) containing the corresponding prices,
times number of times the algorithm is run (the final result is the best of times results, ie. lowest number of clusters found), default is 1,
afriat.par the Afriat parameter, a real number in [0,1], which allows a certain level of error in the optimization of choices; default is 1, ie. no optimization error allowed,
method character string: "fastfloyd" or "floyd" (see Details). Default "fastfloyd".
object an object of class upperBound as returned by cpUpper,
... additional arguments passed to the print and summary methods (unused).

Details

For each run of the algorithm, a random permutation of the observations is drawn, and one by one each observation is associated with the biggest cluster that can include it while preserving consistency with the GARP rationality axiom. If no cluster is compatible with a given observation a new cluster is created to accomodate it.

Three methods are available: "fastfloyd" (default) uses an iterative variant of the Floyd-Warshall algorithm, in which the check of consistency of the current observation with a given cluster is done in a single step of the Floyd-Warshall algorithm. Much faster than "floyd".
"floyd" uses the algorithm described in Crawford and Pendakur (2013): at each step the complete Floyd-Warshall algorithm is run to check whether each cluster can accomodate the current observation.

Value

cpUpper returns an object of class upperBound which contains the following elements:

clustering numeric vector with length equal to number of observations, containing the cluster number of each observation,
cluster.pop numeric vector containing the numbers of observations allocated to each cluster,
hist.n.types numeric vector containing the history of numbers of clusters found during multiple runs of the algorithm.
n.types upper bound on the number of types,
afriat.par Afriat parameter used in the algorithm.

Note

Warning: cpUpper can be very slow for large datasets (eg. more than 1000 observations).

Author(s)

Julien Boelaert <jubo.stats@gmail.com>
directPrefs

References

See Also
See cpLower for the lower bound on number of types.

Examples
# Cluster GARP-violating data:
data(noGarp)
cp.up <- cpUpper(noGarp$x, noGarp$p)
cp.up$clustering

directPrefs Compute direct and indirect revealed preferences.

Description
The directPrefs function computes direct revealed preferences, the indirectPrefs function computes indirect revealed preferences.

Usage
directPrefs(x, p, afriat.par = 1)
indirectPrefs(x, p, afriat.par = 1)

Arguments
x data frame or matrix containing the observed quantities, where each row corresponds to an observation and the columns are types of goods.
p data frame or matrix (of same dimensions as x) containing the corresponding prices.
afriat.par the Afriat parameter, real number in [0,1], which allows a certain level of error in the optimization of choices; default is 1, ie. no optimization error allowed.

Details
Direct preferences are directly computed from matrix multiplication of prices and quantities. Indirect preferences are computed using a variant of the Floyd-Warshall algorithm.

Value
A matrix of direct or indirect revealed preferences:
prefs[i, j] = 0 if bundle i is not revealed prefered to bundle j
prefs[i, j] = 1 if bundle i is revealed prefered to bundle j
prefs[i, j] = 2 if bundle i is strictly revealed prefered to bundle j
Author(s)

Julien Boelaert <jubo.stats@gmail.com>

References


Examples

# Compute direct and indirect preferences of SARP-violating data:
data(noSarp)
directPrefs(noSarp$x, noSarp$p)
indirectPrefs(noSarp$x, noSarp$p)

Description

Toy examples to test for consistency with rationality axioms.

Usage

data(noGarp)
data(noWarp)
data(noSarp)
data(okSarp)
data(noAxiom)

Format

Each dataset is a list of two matrices, named x and p, that contain quantities and prices, respectively. Each row is an observation, each column is a type of good.

Details

Data in noAxiom violate WARP, GARP and SARP.
Data in noWarp violate WARP and SARP, but not GARP.
Data in noGarp violate GARP and SARP, but not WARP.
Data in noSarp violate SARP, but not WARP or GARP.
Data in okSarp are consistent with WARP, SARP and GARP.

Author(s)

Julien Boelaert <jubo.stats@gmail.com>
Examples

```r
data(noWarp)
checkWarp(noWarp$x, noWarp$p)
```

### simGarp

**Generate random data consistent with rationality axioms (WARP, SARP, GARP).**

### Description

Functions for generating random data (prices and quantities) consistent with the chosen rationality axiom.

### Usage

```r
simWarp(nobs, ngoods, afriat.par = 1, maxit = 10 * nobs, qmin = 0, qmax = 1, pmin = 0, pmax = 1)
simSarp(nobs, ngoods, afriat.par = 1, maxit = 10 * nobs, qmin = 0, qmax = 1, pmin = 0, pmax = 1)
simGarp(nobs, ngoods, afriat.par = 1, maxit = 10 * nobs, qmin = 0, qmax = 1, pmin = 0, pmax = 1)
```

### Arguments

- `nobs`: the desired number of observations (number of rows in the quantities and prices matrices),
- `ngoods`: the number of goods in the dataset (number of columns in the quantities and prices matrices),
- `afriat.par`: the Afriat parameter, a real number in [0,1], which allows a certain level of error in the optimization of choices; default is 1, i.e. no optimization error allowed,
- `maxit`: maximum number of iterations (default to 10 times `nobs`),
- `qmin`: minimum quantities for each good,
- `qmax`: maximum quantities for each good,
- `pmin`: minimum prices for each good,
- `pmax`: maximum prices for each good.

### Details

The data are iteratively incremented: at each iteration a new random observation (prices and quantities) is generated, and is accepted only if it is consistent with the previously accepted data, in which case it is added to the data. The random observations (price-quantities couples) are independently generated from uniform distributions in the support defined by `qmin`, `qmax`, and `pmin`, `pmax`.

For GARP and SARP the depth-first search method is used to check for consistency (a recursive search using only the new candidate observation as starting point), for WARP the candidate observation is pairwise checked against all previously accepted data.
The algorithm stops if the desired number of observations $nobs$ is reached. If the desired number of observations $nobs$ is not reached in $maxit$ iterations, a warning is issued and the function returns the largest dataset attained.

**Value**
- $x$: numeric matrix of generated quantities,
- $p$: numeric matrix of generated prices,
- $iter$: number of iterations before the algorithm stopped,
- $nobs$: number of generated observations.

**Author(s)**
Julien Boelaert <jubo.stats@gmail.com>

**References**

**See Also**
See checkGarp for rationality tests.

**Examples**
```r
# Generate 100 WARP-consistent observations of 5 goods
simdata <- simWarp(100, 5)
simPrefs()
```

**Description**
Function for generating random data (prices and quantities) consistent with a given direct preferences matrix.

**Usage**
```
simPrefs(pref.mat, ngoods, afriat.par = 1,
    qmin = 0, qmax = 1, pmin = 0, pmax = 1,
    maxit = 100, verbose = FALSE)
```
Arguments

- pref.mat: the desired matrix of direct preferences (must be a square matrix, see details),
- ngoods: the number of goods in the dataset (number of columns in the quantities and prices matrices),
- afriat.par: the Afriat parameter, a real number in [0,1], which allows a certain level of error in the optimization of choices; default is 1, i.e. no optimization error allowed,
- qmin: minimum quantities for each good,
- qmax: maximum quantities for each good,
- pmin: minimum prices for each good,
- pmax: maximum prices for each good,
- maxit: maximum number of iterations of the optimization algorithm (default to 100),
- verbose: logical value: if TRUE a trace of the optimization algorithm is printed.

Details

The input pref.mat must be a square matrix, with number of rows equal to the number of desired observations. All entries must be either 0, 1, or 2. The interpretation of the matrix is the same as in function directPrefs:

- pref.mat[i,j] = 0 if bundle i is not revealed preferred to bundle j
- pref.mat[i,j] = 1 if bundle i is revealed preferred to bundle j
- pref.mat[i,j] = 2 if bundle i is strictly revealed preferred to bundle j.

All diagonal entries of pref.mat must be 1 (each bundle is revealed preferred to itself), except when afriat.par is strictly less than 1.

The simulated data (quantities and prices) are obtained by particle swarm optimization (of package pso). Fitness must reach 0 for the data to be consistent with the preference matrix. If optimization fails, a warning is issued.

Value

- x: numeric matrix of generated quantities, or NULL if optimization fails,
- p: numeric matrix of generated prices, or NULL if optimization fails.

Author(s)

Julien Boelaert <jubo.stats@gmail.com>

References


See Also

See directPrefs for computation of preferences.
Examples

# Generate 3 observations of 5 goods, according to a given preference matrix
pref.mat <- matrix(c(1,0,0,2,1,0,2,2,1), 3)
simdata <- simPrefs(pref.mat = pref.mat, ngoods = 5)
Index

*Topic datasets
  noGarp, 9

*Topic methods
  checkGarp, 3
  cpLower, 5
  cpUpper, 6

*Topic package
  revealedPrefs-package, 2

checkGarp, 2, 3, 11
checkSarp (checkGarp), 3
checkWarp (checkGarp), 3
cpLower, 5, 8
cpUpper, 2, 6, 6

directPrefs, 2, 8, 12
indirectPrefs (directPrefs), 8

noAxiom (noGarp), 9
noGarp, 9
noSarp (noGarp), 9
noWarp (noGarp), 9

okSarp (noGarp), 9

print.axiomTest (checkGarp), 3
print.lowerBound (cpLower), 5
print.upperBound (cpUpper), 6

revealedPrefs (revealedPrefs-package), 2
revealedPrefs-package, 2

simGarp, 10
simPrefs, 11
simSarp (simGarp), 10
simWarp (simGarp), 10
summary.axiomTest (checkGarp), 3
summary.lowerBound (cpLower), 5
summary.upperBound (cpUpper), 6