Package 'matchingMarkets'

December 4, 2018

Version 1.0-0

Depends R (>= 3.0.2)

Imports Rcpp (>= 0.12.12), RcppProgress (>= 0.2), rJava, lpSolve, lattice, partitions, parallel, stats, utils

SystemRequirements Java, C++11

LinkingTo Rcpp, RcppProgress, RcppArmadillo

Suggests knitr, ggplot2, grDevices, graphics

VignetteBuilder knitr

Title Analysis of Stable Matchings

Author Thilo Klein [aut, cre],
Sven Giegerich [ctb],
Alexander Sauer [ctb]

Maintainer Thilo Klein <thilo@klein.uk>

Description Implements structural estimators to correct for the sample selection bias from observed outcomes in matching markets. This includes one-sided matching of agents into groups as well as two-sided matching of students to schools. The package also contains algorithms to find stable matchings in the three most common matching problems: the stable roommates problem, the college admissions problem, and the house allocation problem.

URL http://matchingMarkets.org, http://klein.uk

BugReports https://github.com/thiloklein/matchingMarkets/issues

License GPL (>= 2)

RoxygenNote 6.1.0

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-12-04 10:20:14 UTC

R topics documented:

`matchingMarkets-package`

An R package for the analysis of stable matchings.

Description

The `matchingMarkets` package contains R, `C++` and `Java` code for stable matching algorithms and the estimation of structural models that correct for the sample selection bias of observed outcomes in matching markets.

Matching is concerned with who transacts with whom, and how. For example, who works at which job, which students go to which school, who forms a workgroup with whom, and so on.

The empirical analysis of matching markets is naturally subject to sample selection problems. If agents match assortatively on characteristics unobserved to the analyst but correlated with both the exogenous variable and the outcome of interest, regression estimates will generally be biased.

The `matchingMarkets` package comprises

1. *Bayes estimators*. The estimators implemented in function `stabit` and `stabit2` correct for the selection bias from endogenous matching.

   The current package version provides solutions for two commonly observed matching processes: (i) the *group formation problem* with fixed group sizes and (ii) the *college admissions problem*. These processes determine which matches are observed – and which are not – and this is a sample selection problem.

2. *Post-estimation tools*. Setting `mfx=TRUE` in the `summary` function computes marginal effects from coefficients in binary outcome and selection equations and `khb` implements the Karlson-Holm-Breen test for confounding due to sample selection.
3. *Design matrix generation*. The estimators are based on independent variables for all feasible, i.e., observed and counterfactual, matches in the market. Generating the characteristics of all feasible matches from individual-level data is a combinatorial problem. The package returns design matrices based on pre-specified transformations to generate counterfactual matches.
4. *Algorithms*. The package also contains matching algorithms that can be used to simulated matching data: `hri`: A constraint model (Posser, 2014) for the stable marriage and college admissions problem, a.k.a. hospital/residents problem (see Gale and Shapley, 1962). `sri`: A constraint model for the stable roommates problem (see Gusfield and Irving, 1989). `ttc`: The top-trading-cycles algorithm for the housing market problem. These can be used to obtain stable matchings from simulated or real preference data (see Shapley and Scarf, 1974).
5. *Data*. In addition to the `baac00` dataset from borrowing groups in Thailands largest agricultural lending program, the package provides functions `stabsim` and `stabsim2` to simulate one's own data from one-sided and two-sided matching markets.

Frequently Asked Questions

- *Why can I not use the classic Heckman correction?*

  Estimators such as the Heckman (1979) correction (in package `sampleSelection`) or double selection models are inappropriate for this class of selection problems. To see this, note that a simple first stage discrete choice model assumes that an observed match reveals match partners' preferences over each other. In a matching market, however, agents can only choose from the set of partners who would be willing to form a match with them and we do not observe the players' relevant choice sets.

- *Do I need an instrumental variable to estimate the model?*

  Short answer: No. Long answer: The characteristics of other agents in the market serve as the source of exogenous variation necessary to identify the model. The identifying exclusion restriction is that characteristics of all agents in the market affect the matching, i.e., who matches with whom, but it is only the characteristics of the match partners that affect the outcome of a particular match once it is formed. No additional instruments are required for identification (Sorensen, 2007).

- *What are the main assumptions underlying the estimator?*

  The approach has certain limitations rooted in its restrictive economic assumptions.

  1. The matching models are *complete information* models. That is, agents are assumed to have a complete knowledge of the qualities of other market participants.
  2. The models are *static equilibrium* models. This implies that (i) the observed matching must be an equilibrium, i.e., no two agents would prefer to leave their current partners in order to form a new match (definition of pairwise stability), and (ii) the equilibrium must be unique for the likelihood function of the model to be well defined (Bresnahan and Reiss, 1991).
  3. Uniqueness results can be obtained in two ways. First, as is common in the industrial organization literature, by imposing suitable *preference restrictions*. A suitable restriction on agents' preferences that guarantees a unique equilibrium is alignment (Pycia, 2012). In a group formation model, (pairwise) preference alignment states that any two agents

who belong to the same groups must prefer the same group over the other. Second, by choosing a *market assigment* based on matching algorithms that produce a unique stable matching, such as the well-studied Gale and Shapley (1962) deferred acceptance algorithm.

4. Finally, the models assume *bivariate normality* of the errors in selection and outcome equation. If that assumption fails, the estimator is generally inconsistent and can provide misleading inference in small samples.

How to cite this package

Whenever using this package, please cite as

Klein, T. (2018). matchingMarkets: Structural Estimator and Algorithms for the Analysis of Stable Matchings. R package version 1.0-0.

Author(s)

Thilo Klein

References

Bresnahan, T. and Reiss, P. (1991). Empirical models of discrete games. *Journal of Econometrics*, 48(1-2):57–81.

Gale, D. and Shapley, L.S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15.

Gusfield, D.M. and R.W. Irving (1989). The stable marriage problem: Structure and algorithms, MIT Press.

Heckman, J. (1979). Sample selection bias as a specification error. *Econometrica*, 47(1):153–161.

Prosser, P. (2014). Stable Roommates and Constraint Programming. *Lecture Notes in Computer Science, CPAIOR 2014 Edition*. Springer International Publishing, 8451: 15–28.

Pycia, M. (2012). Stability and preference alignment in matching and coalition formation. *Econometrica*, 80(1):323–362.

Shapley, L. and H. Scarf (1974). On cores and indivisibility. *Journal of Mathematical Economics*, 1(1):23–37.

Sorensen, M. (2007). How smart is smart money? A two-sided matching model of venture capital. *The Journal of Finance*, 62(6):2725–2762.

See Also

sampleSelection

baac00 *Townsend Thai Project BAAC Annual Resurvey, 2000*

Description

The baac00 data frame contains data of 292 borrowers from Thailand's largest agricultural lending program. These data are collected as part of the Townsend Thai Project Bank for Agriculture and Agricultural Cooperatives (BAAC) Annual Resurvey (Townsend, 2000). The 292 borrowers are nested within 68 groups and 39 markets. This nestedness makes the dataset particularly relevant for matching applications. A more complete discussion of the data is found in Ahlin (2009), Section 3, and Klein (2015a).

Usage

```
data(baac00)
```

Format

This data frame contains the following columns:

g.id group identifier.

m.id market identifier.

R repayment outcome: BAAC never raised interest rate as a penalty for late repayment.

pi success probability: measure of group members' project success probability.

wst worst year: indicator of economically worst year. 1:last year; 2:year before last year; 101-168:neither.

loan_size loan size: average loan size borrowed by the group.

loan_size2 loan size squared.

lngroup_agei log group age: log of number of years group has existed.

Source

Townsend, R. (2000). Townsend Thai Project Bank for Agriculture and Agricultural Cooperatives (BAAC) Annual Resurvey, 2000. Available at http://hdl.handle.net/1902.1/12057, *Murray Research Archive*.

References

Ahlin, C. (2009). Matching for credit: Risk and diversification in Thai microcredit groups. Working Paper 251, *Bureau for Research and Economic Analysis of Development*.

Klein, T. (2015a). Does Anti-Diversification Pay? A One-Sided Matching Model of Microcredit. *Cambridge Working Papers in Economics*, #1521.

`hri` *All stable matchings in the hospital/residents problem with incomplete lists*

Description

Finds *all* stable matchings in either the hospital/residents problem (a.k.a. college admissions problem) or the related stable marriage problem. Dependent on the problem, the results comprise the student and college-optimal or the men and women-optimal matchings. The implementation allows for *incomplete preference lists* (some agents find certain agents unacceptable) and *unbalanced instances* (unequal number of agents on both sides). The function uses the Prosser (2014) constraint encoding based on either given or randomly generated preferences.

Usage

```
hri(nStudents = ncol(s.prefs), nColleges = ncol(c.prefs),
  nSlots = rep(1, nColleges), s.prefs = NULL, c.prefs = NULL,
  s.range = NULL, c.range = NULL, randomization = NULL,
  seed = NULL, check_consistency = TRUE, ...)
```

Arguments

`nStudents`	integer indicating the number of students (in the college admissions problem) or men (in the stable marriage problem) in the market. Defaults to `ncol(s.prefs)`.
`nColleges`	integer indicating the number of colleges (in the college admissions problem) or women (in the stable marriage problem) in the market. Defaults to `ncol(c.prefs)`.
`nSlots`	vector of length `nColleges` indicating the number of places (i.e. quota) of each college. Defaults to `rep(1,nColleges)` for the marriage problem.
`s.prefs`	matrix of dimension `nColleges x nStudents` with the `j`th column containing student `j`'s ranking over colleges in decreasing order of preference (i.e. most preferred first).
`c.prefs`	matrix of dimension `nStudents x nColleges` with the `i`th column containing college `i`'s ranking over students in decreasing order of preference (i.e. most preferred first).
`s.range`	range of two intergers `s.range = c(s.min, s.max)`, where `s.min < s.max`. Produces incomplete preference lists with the length of each student's list randomly sampled from the range `[s.min, s.max]`. Note: interval is only correct if either c.range or s.range is used.
`c.range`	range of two intergers `c.range = c(c.min, c.max)`, where `c.min < c.max`. Produces incomplete preference lists with the length of each college's list randomly sampled from the range `[c.min, c.max]`. Note: interval is only correct if either c.range or s.range is used.
`randomization`	determines at which level random lottery numbers for student priorities are drawn. The default is `randomization = "multiple"`, where a student's priority is determined by a separate lottery at each college (i.e. local tie-breaking).

For the second variant, `randomization = "single"`, a single lottery number determines a student's priority at all colleges (i.e. global tie breaking).

`seed` integer setting the state for random number generation.

`check_consistency`
Performs consicentcy checks (Checks if there are columns in the preference matrices that only contains zeros and drops them and checks the matrixes for consistencies if they are given by characters). Defaults to `TRUE` but changing it to `FALSE` might reduce the running-time for large problems.

`...` .

Value

`hri` returns a list of the following elements.

`s.prefs.smi` student-side preference matrix for the stable marriage problem with incomplete lists (SMI).

`c.prefs.smi` college-side preference matrix for the stable marriage problem with incomplete lists (SMI).

`s.prefs.hri` student-side preference matrix for the college admissions problem (a.k.a. hospital/residents problem) with incomplete lists (HRI).

`c.prefs.hri` college-side preference matrix for the college admissions problem (a.k.a. hospital/residents problem) with incomplete lists (HRI).

`matchings` edgelist of matched students and colleges, inculding the number of the match (`matching`) and two variables that indicate the student-optimal match (`sOptimal`) and college-optimal match (`cOptimal`)

.

Minimum required arguments

`hri` requires the following combination of arguments, subject to the matching problem.

`nStudents, nColleges` Marriage problem with random preferences.

`s.prefs, c.prefs` Marriage problem with given preferences.

`nStudents, nSlots` College admissions problem with random preferences.

`s.prefs, c.prefs, nSlots` College admissions problem with given preferences.

Author(s)

Thilo Klein

References

Gale, D. and L.S. Shapley (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15.

Morizumi, Y., T. Hayashi and Y. Ishida (2011). A network visualization of stable matching in the stable marriage problem. *Artificial Life Robotics*, 16:40–43.

Prosser, P. (2014). Stable Roommates and Constraint Programming. *Lecture Notes in Computer Science, CPAIOR 2014 Edition*. Springer International Publishing, 8451: 15–28.

Examples

```
## Not run:
## ----------------------
## --- Marriage problem

## 7 men, 6 women, random preferences:
 hri(nStudents=7, nColleges=6, seed=4)

## 3 men, 2 women, given preferences:
 s.prefs <- matrix(c(1,2, 1,2, 1,2), 2,3)
 c.prefs <- matrix(c(1,2,3, 1,2,3), 3,2)
 hri(s.prefs=s.prefs, c.prefs=c.prefs)

## 3 men, 2 women, given preferences:
 s.prefs <- matrix(c("x","y", "x","y", "x","y"), 2,3)
 colnames(s.prefs) <- c("A","B","C")
 c.prefs <- matrix(c("A","B","C", "A","B","C"), 3,2)
 colnames(c.prefs) <- c("x","y")
 hri(s.prefs=s.prefs, c.prefs=c.prefs)

## --------------------------------
## --- College admission problem

## 7 students, 2 colleges with 3 slots each, random preferences:
 hri(nStudents=7, nSlots=c(3,3), seed=21)

## 7 students, 2 colleges with 3 slots each, given preferences:
 s.prefs <- matrix(c(1,2, 1,2, 1,NA, 1,2, 1,2, 1,2, 1,2), 2,7)
 c.prefs <- matrix(c(1,2,3,4,5,6,7, 1,2,3,4,5,NA,NA), 7,2)
 hri(s.prefs=s.prefs, c.prefs=c.prefs, nSlots=c(3,3))

## 7 students, 2 colleges with 3 slots each, given preferences:
 s.prefs <- matrix(c("x","y", "x","y", "x",NA, "x","y",
                     "x","y", "x","y", "x","y"), 2,7)
 colnames(s.prefs) <- c("A","B","C","D","E","F","G")
 c.prefs <- matrix(c("A","B","C","D","E","F","G",
                     "A","B","C","D","E",NA,NA), 7,2)
 colnames(c.prefs) <- c("x","y")
 hri(s.prefs=s.prefs, c.prefs=c.prefs, nSlots=c(3,3))

## 7 students, 3 colleges with 3 slots each, incomplete preferences:
 hri(nStudents=7, nSlots=c(3,3,3), seed=21, s.range=c(1,3))

 s.prefs <- matrix(c('S1', 'S2', NA,
                     'S3', 'S1', NA,
                     'S1', NA, NA,
                      NA, NA,NA,
                     'S2', 'S1', 'S3'),
                   nrow = 3, ncol = 5)
 colnames(s.prefs) <- c('A', 'B', 'C', 'D', 'E')
 c.prefs <- matrix(c('B', 'C','D', 'A',
                     'C', 'D', NA, NA,
```

```
                    'D', 'B', 'A', 'E'),
                  nrow = 4, ncol = 3)
 colnames(c.prefs) <- c('S1', 'S2', 'S3')
 hri(s.prefs=s.prefs, c.prefs=c.prefs, nSlots=c(3,3,3), check_consistency = TRUE)
## --------------------
## --- Summary plots

## 200 students, 200 colleges with 1 slot each
 res <- hri(nStudents=200, nColleges=200, seed=12)
 plot(res)
 plot(res, energy=TRUE)

## End(Not run)
```

`hri2` *Resident-optimal matching in the hospital/residents problem with couples*

Description

Implements the Roth Peranson matching algorithm for the hospital/residents problem with couples as described in Roth and Peranson (1999). The function is based on an adoption of Bacchus (2018).

Usage

```
hri2(nStudents = ncol(s.prefs), nColleges = ncol(c.prefs),
  nSlots = rep(1, nColleges), nCouples = ncol(co.prefs),
  s.prefs = NULL, c.prefs = NULL, co.prefs = NULL,
  randomization = "multiple", seed = NULL, check_consistency = TRUE,
  ...)
```

Arguments

`nStudents`	integer indicating the number of students (in the college admissions problem) or men (in the stable marriage problem) in the market. Defaults to `ncol(s.prefs)`.
`nColleges`	integer indicating the number of colleges (in the college admissions problem) or women (in the stable marriage problem) in the market. Defaults to `ncol(c.prefs)`.
`nSlots`	vector of length `nColleges` indicating the number of places (i.e. quota) of each college. Defaults to `rep(1,nColleges)` for the marriage problem.
`nCouples`	integer indicating the number of couples (in the college admissions problem) or men (in the stable marriage problem) in the market. Defaults to `ncol(co.prefs)`
`s.prefs`	matrix of dimension `nColleges` x `nStudents` with the `j`th column containing student `j`'s ranking over colleges in decreasing order of preference (i.e. most preferred first).
`c.prefs`	matrix of dimension `nStudents` x `nColleges` with the `i`th column containing college `i`'s ranking over students in decreasing order of preference (i.e. most preferred first).

`co.prefs` matrix of dimension `4 x nCouplesPrefs` in long format with the `1`th and `2`th columns containing student couple id's; `3`th and `4`th is a 2-tuple ranking over college preference for the couple (coupleStudent1.pref, coupleStudent2.pref) in decreasing order of preference by rows (i.e. most preferred first).

`randomization` determines at which level and in which order random lottery numbers for student priorities are drawn. The default is `randomization = "multiple"`, where a student's priority is determined by a separate lottery at each college (i.e. local tie-breaking). For the second variant, `randomization = "single"`, a single lottery number determines a student's priority at all colleges (i.e. global tie breaking). A third variant is common in the context of course allocation, where a "couple" represents a student who submits a preference ranking over single courses (first course) and combinations of courses (first and second course). Here, the option `randomization = "single-course-first"` gives applications for a student's single courses strictly higher priority than for course combinations. This ensures the fairness criterion that a student is only assigned a second course after single course applications of all students have been considered.

`seed` integer setting the state for random number generation.

`check_consistency`
Performs additional consicentcy checks if the preference matrices are given by characters. Defaults to `FALSE`. Set to `FALSE` to reduce run-time.

`...` .

Value

`hri2` returns a list of the following elements:

`matchings` List of matched students and colleges.

`summary` Detailed report of the matching result, including futher information on ranks.

Minimum required arguments

`hri2` requires the following combination of arguments, subject to the matching problem.

`nStudents, nColleges` Residence hospital problem without couples and random preferences

`nStudents, nColleges, nCouples, nSlots` Residence hospital problem with couples and random preferences.

`s.prefs, c.prefs, co.prefs, nSlots` Residence hospital problem with couples and given preferences.

Author(s)

Sven Giegerich, Thilo Klein

References

Bacchus, F. (2018). Stable matching suite. GitHub repository.

Gale, D. and L.S. Shapley (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15.

Roth, A. E., & Peranson, E. (1999). The redesign of the matching market for American physicians: Some engineering aspects of economic design. *American economic review*, 89(4), 748-780.

Kojima, F., Pathak, P. A., & Roth, A. E. (2013). Matching with couples: Stability and incentives in large markets. *The Quarterly Journal of Economics*, 128(4), 1585-1632.

Examples

```
## Not run:
## Example with given preferences
(s.prefs <- matrix(c(4,2,3,5, 2,1,3,NA, 1,2,3,4), 4,3))
(c.prefs <- matrix(rep(1:5,5), 5,5))
(co.prefs <- matrix(c(rep(4,3), rep(5,3), 3,3,NA, 3,NA,3), 3,4))
res <- hri2(s.prefs=s.prefs, c.prefs=c.prefs, co.prefs=co.prefs, nSlots=rep(1,5))
res$matchings
# summary(res)

## Example with random preferences
nStudents <- 50
nColleges <- 30
nCouples <- 4
nSlots <- sample(1:nStudents, nColleges)
res <- hri2(nStudents=nStudents, nColleges=nColleges, nCouples=nCouples, nSlots=nSlots)
res$matchings
# summary(res)

## Example with characters in the preferences matrices
s.prefs <- matrix(c("Micro1", NA, NA,
                    "Micro2", "Micro1", "Macro",
                    "Macro",NA ,NA),
                    ncol = 3)
colnames(s.prefs) <- c('Lea', 'Mia', 'Kai')
c.prefs <- matrix(c("Niklas", "Kai", "Mia", "Anna",
                    "Lea", "Kai", "Anna",NA,
                    "Kai", "Mia", "Lea",NA),
                    ncol = 3)
colnames(c.prefs) <- c('Micro1', 'Micro2', 'Macro')
col1 <- c(rep("Niklas",4),rep("Anna",5))
col2 <- c(rep("Jan",4),rep("Lisa",5))
col3 <- c("Micro1","Macro","Micro1",NA,"Macro",
          NA,"Micro2","Micro2","Macro")
col4 <- c("Micro2","Micro1",NA,"Macro","Macro",
          "Micro1","Micro2","Macro",NA)
co.prefs <- matrix(c(col1,col2,col3,col4), ncol = 4)
res <- hri2(s.prefs=s.prefs, c.prefs=c.prefs, co.prefs=co.prefs,
            nSlots=c(2,1,1))
res$matching
```

```
## Example if students are allowed to apply and be accepted by two courses
col12 <- c(rep(c(rep("Niklas",4),rep("Anna",2)),2))
col3 <- c("Micro1","Macro","Micro1","Macro","Macro","Macro")
col4 <- c("Micro2","Micro1",NA,NA,"Micro1","Micro2")
co.prefs <- matrix(c(col12,col3,col4), ncol = 4)
res <- hri2(s.prefs=s.prefs, c.prefs=c.prefs, co.prefs=co.prefs,
            nSlots=c(2,1,1))
res$matching

## End(Not run)
```

`iaa` *Immediate Acceptance Algorithm (a.k.a. Boston mechanism) for two-sided matching markets*

Description

Finds the optimal assignment of students to colleges in the college admissions problem based on the Boston mechanism. The algorithmen is also applicable to the stable marriage problem. The option `acceptance="deferred"` instead uses the Gale-Shapley (1962) Deferred Acceptance Algorithm with student offer. The function works with either given or randomly generated preferences.

Usage

```
iaa(nStudents = ncol(s.prefs), nColleges = ncol(c.prefs),
  nSlots = rep(1, nColleges), s.prefs = NULL, c.prefs = NULL,
  acceptance = "immediate", short_match = TRUE, seed = 123)
```

Arguments

`nStudents`	integer indicating the number of students (in the college admissions problem) or men (in the stable marriage problem) in the market. Defaults to `ncol(s.prefs)`.
`nColleges`	integer indicating the number of colleges (in the college admissions problem) or women (in the stable marriage problem) in the market. Defaults to `ncol(c.prefs)`.
`nSlots`	vector of length `nColleges` indicating the number of places (i.e. quota) of each college. Defaults to `rep(1,nColleges)` for the marriage problem.
`s.prefs`	matrix of dimension `nColleges x nStudents` with the `j`th column containing student `j`'s ranking over colleges in decreasing order of preference (i.e. most preferred first).
`c.prefs`	matrix of dimension `nStudents x nColleges` with the `i`th column containing college `i`'s ranking over students in decreasing order of preference (i.e. most preferred first).
`acceptance`	if `acceptance="deferred"` returns the solution found by the student-proposing Gale-Shapley deferred acceptance algorithm; if `acceptance="immediate"` (the default) returns the solution found by the Boston mechanism.

short_match	(Optional) If FALSE then in the returned matching, free capacities will be indicated with 0 entries. If TRUE, free capacities will not be reported in the returned matching but an additonal data.frame is returned that contains free capacities. Defaults to TRUE.
seed	(Optional) integer setting the state for random number generation.

Value

iaa returns a list with the following elements.

s.prefs	student-side preference matrix.
c.prefs	college-side preference matrix.
iterations	number of interations required to find the stable matching.
matchings	edgelist of matches
singles	identifier of single (or unmatched) students/men.

Minimum required arguments

iaa requires the following combination of arguments, subject to the matching problem.

nStudents, nColleges Marriage problem with random preferences.

s.prefs, c.prefs Marriage problem with given preferences.

nStudents, nSlots College admissions problem with random preferences.

s.prefs, c.prefs, nSlots College admissions problem with given preferences.

Author(s)

Thilo Klein

References

Gale, D. and Shapley, L.S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15.

Kojima, F. and M.U. Unver (2014). The "Boston" school-choice mechanism. *Economic Theory*, 55(3): 515–544.

Examples

```
## Not run:
## --------------------------------
## --- College admission problem

s.prefs <- matrix(c(1,2,3,
                    1,2,3,
                    1,2,3,
                    2,1,3,
                    2,1,3),
                  byrow = FALSE, ncol = 5, nrow = 3); s.prefs
```

```
c.prefs <- matrix(c(1,4,2,3,5,
                    5,2,3,4,1,
                    1,2,3,4,5),
                  byrow = FALSE, ncol = 3, nrow = 5); c.prefs
nSlots <- c(2,2,1)

## Boston mechanism
 iaa(s.prefs = s.prefs, c.prefs = c.prefs, nSlots = nSlots)$matchings

## Gale-Shapley algorithm
 iaa(s.prefs = s.prefs, c.prefs = c.prefs, nSlots = nSlots, acceptance="deferred")$matchings

## Same results for the Gale-Shapley algorithm with hri2() function (but different format)
 set.seed(123)
 iaa(nStudents=7, nSlots=c(3,3), acceptance="deferred")$matchings
 set.seed(123)
 hri2(nStudents=7, nSlots=c(3,3))$matchings

## End(Not run)
```

---

khb *Karlson-Holm-Breen method for comparing probit coefficients*

Description

Significance test for confounding; that is, the difference between regression coefficients from same-sample nested logit and probit models. The test procedure follows Karlson et al (2012), Section 3.4.

Usage

```
khb(X, y, z)
```

Arguments

X	data frame comprising independent variables including confounding variable.
y	vector of dependent variable.
z	character string giving the name of the confounding variable in X.

Author(s)

Thilo Klein

References

Karlson, K.B., A. Holm and R. Breen (2012). Comparing regression coefficients between same-sample nested models using logit and probit: A new method. *Sociological Methodology*, 42(1):286–313.

Examples

```
## 1. load results from Klein (2015a)
 data(klein15a)

## 2. apply KHB method
with(klein15a$variables, khb(X=X, y=Y, z="eta"))
```

`klein15a` *MCMC results in Klein (2015a)*

Description

MCMC results in Klein (2015a).

Usage

```
data(klein15a)
```

Format

A list containing the following elements:

model.list .

coefs .

References

Klein, T. (2015a). Does Anti-Diversification Pay? A One-Sided Matching Model of Microcredit. *Cambridge Working Papers in Economics*, #1521.

`klein15b` *Results of Monte Carlo Simulations in Klein (2015b)*

Description

Results of Monte Carlo Simulations in Klein (2015b) for 40 two-group markets.

Usage

```
data(klein15b)
```

Format

A list containing the following elements:

exp.5.5.ols Benchmark study, OLS: coefficient estimates for 40 markets with groups of 5. Data for all 5 group members is observed.

exp.5.5.ntu Benchmark study, structural model.

exp.6.5.ols Experiment 1, OLS: coefficient estimates for 40 markets with groups of 6. Only Data for 5 group members is observed.

exp.6.5.ntu Experiment 1, structural model.

exp.6.6.ols Experiment 2, OLS: coefficient estimates for 40 markets with groups of 6. Data for all 6 group members is observed but only a random sample of 250 of the 922 counterfactual groups is used in the analysis.

exp.6.6.ntu Experiment 2, structural model.

References

Klein, T. (2015a). Does Anti-Diversification Pay? A One-Sided Matching Model of Microcredit. *Cambridge Working Papers in Economics*, #1521.

Klein, T. (2015b). Analysis of stable matchings in R: Package matchingMarkets. *Vignette to R package matchingMarkets*, The Comprehensive R Archive Network.

Examples

```
## Plot of posterior distributions

data(klein15b)

tpe <- c(rep("Benchmark",2), rep("Experiment 1",2), rep("Experiment 2",2))

for(i in seq(1,length(klein15b)-1,2)){
  ntu <- klein15b[[i]]
  ols <- klein15b[[i+1]]

  ntu <- ntu[,colnames(ntu) == "beta.wst.ieq"]
  ols <- ols[,colnames(ols) == "beta.wst.ieq"]

  if(i == 1){
    draws <- data.frame(Structural=ntu, OLS=ols, type=tpe[i]) #, stringsAsFactors=FALSE
  } else{
    draws <- rbind(draws, data.frame(Structural=ntu, OLS=ols, type=tpe[i]))
  }
}

library(lattice)
lattice.options(default.theme = standard.theme(color = FALSE))
keys <- list(text=c("Structural model","OLS"), space="top", columns=2, lines=TRUE)
densityplot( ~ Structural + OLS | type, plot.points=FALSE, auto.key=keys,
       data = draws, xlab = "coefficient draws", ylab = "density", type = "l",
       panel = function(x,...) {
```

```
          panel.densityplot(x,...)
          panel.abline(v=-1, lty=3)
        })

## Not run:
## Modes of posterior distributions

## load data
data(klein15b)

## define function to obtain the mode
mode <- function(x){
  d <- density(x,bw="SJ")
  formatC(round(d$x[which.max(d$y)], 3), format='f', digits=3)
}

## Benchmark study
apply(klein15b$exp.5.5.ntu, 2, mode)
apply(klein15b$exp.5.5.ols, 2, mode)

## Experiment 1
apply(klein15b$exp.6.5.ntu, 2, mode)
apply(klein15b$exp.6.5.ols, 2, mode)

## Experiment 2
apply(klein15b$exp.6.6.ntu, 2, mode)
apply(klein15b$exp.6.6.ols, 2, mode)

## End(Not run)
```

---

mce *MC Experiments*

Description

MC Experiments

Usage

```
mce(seed, niter, N, n, m, type, method)
```

Arguments

seed	seed value for Monte Carlo Experiment
niter	number of draws in estimation
N	group size (population)
n	group size (sample)
m	number of markets

type	type of the MC Experiment. Either group.members for randomly sampled group members or counterfactual.groups for randomly sampled number of counterfactual (or feasible) groups in selection equation (capped at limit max.combs=250)
method	either group.members or counterfactual.groups

Author(s)

Thilo Klein

Examples

```
## Not run:
## 1. Set parameters
mciter <- 2 #500
niter <- 10 #400000
nodes <- 4

## 2. Setup parallel backend to use 4 processors
library(foreach); library(doSNOW)
cl <- makeCluster(4); registerDoSNOW(cl)

## 3. Define foreach loop function
mce.add <- function(mciter, niter, N, n, m, type, method){
  h <- foreach(i=1:mciter) %dopar% {
    library(matchingMarkets)
    mce(seed=i,niter, N, n, m, type, method)
  }
  do.call(rbind, h)
}

## 4. Run siumlations:

## 4-a. Benchmark study
exp.5.5.ols <- mce.add(mciter=mciter, niter=niter, N=5, n=5, m=40,
                       type="group.members", method="outcome")
exp.5.5.ntu <- mce.add(mciter=mciter, niter=niter, N=5, n=5, m=40,
                       type="group.members", method="NTU")

## 4-b. Experiment 1: randomly sampled group members
exp.6.5.ols <- mce.add(mciter=mciter, niter=niter, N=6, n=5, m=40,
                       type="group.members", method="outcome")
exp.6.5.ntu <- mce.add(mciter=mciter, niter=niter, N=6, n=5, m=40,
                       type="group.members", method="NTU")

## 4-c. Experiment 2: randomly sampled counterfactual groups
exp.6.6.ols <- mce.add(mciter=mciter, niter=niter, N=6, n=6, m=40,
                       type="counterfactual.groups", method="outcome")
exp.6.6.ntu <- mce.add(mciter=mciter, niter=niter, N=6, n=6, m=40,
                       type="counterfactual.groups", method="NTU")

## 5. Stop parallel backend
stopCluster(cl)
```

```
## End(Not run)
```

`plp` *Partitioning Linear Programme for the stable roommates problem*

Description

Finds the stable matching in the stable roommates problem with transferable utility. Uses the Partitioning Linear Programme formulated in Quint (1991).

Usage

```
plp(V = NULL, N = NULL)
```

Arguments

`V`	valuation matrix of dimension `NxN` that gives row-players valuation over column players (or vice versa).
`N`	integer (divisible by 2) that gives the number of players in the market.

Value

`plp` returns a list with the following items.

`Valuation.matrix`
: input values of V.

`Assignment.matrix`
: upper triangular matrix of dimension `NxN` with entries of 1 for equilibrium pairs and 0 otherwise.

`Equilibrium.groups`
: matrix that gives the `N/2` equilibrium pairs and equilibrium partners' mutual valuations.

Author(s)

Thilo Klein

References

Quint, T. (1991). Necessary and sufficient conditions for balancedness in partitioning games. *Mathematical Social Sciences*, 22(1):87–91.

Examples

```
## Roommate problem with 10 players, transferable utility and random preferences:
plp(N=10)

## Roommate problem with 10 players, transferable utility and given preferences:
V <- matrix(rep(1:10, 10), 10, 10)
plp(V=V)
```

`predict.stabit2` *Predict method for fitted matching models*

Description

Calculate predicted values for matching models fitted with functions `stabit` and `stabit2`.

Usage

```
## S3 method for class 'stabit2'
predict(object, newdata = NULL, ...)
```

Arguments

`object`	a fitted object of class `stabit`
`newdata`	optionally, a data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors or the fitted response values are returned.
`...`	.

Author(s)

Thilo Klein

References

Klein, T. (2015a). Does Anti-Diversification Pay? A One-Sided Matching Model of Microcredit. *Cambridge Working Papers in Economics*, #1521.

Examples

```
## load the results from Klein (2015) paper
 data(klein15a)

## predict the latent outcome variable
 predict(klein15a)
```

rsd *Random serial dictatorship mechanism*

Description

Implements the random serial dictatorship algorithm algorithm for a fair division of indivisible objects among individuals. The mechanism takes individuals' prioirty order as an input or alternatively draws a random permutation of the agents form the uniform distribution. Individuals are then successively assigned an object in that order (so the first agent in the ordering gets the first pick and so on).

Usage

```
rsd(nIndividuals = ncol(prefs), nObjects = nrow(prefs), prefs,
  priority, seed = 123, nSlots = rep(1, nObjects))
```

Arguments

`nIndividuals`	integer indicating the number of individuals in the matching problem. Defaults to ncol(prefs).
`nObjects`	integer indication the number of objects in the matching problem. Defaults to nrow(prefs).
`prefs`	matrix of dimension `nObjects` x `nIndividuals` with the jth column containing students j's ranking over the objects in decreasing order of preference (i.e. most preferred first).
`priority`	(Optional) vector of length `nIndividuals` indicating the priority of the individuals in decreasing order (i.e. highest individuals first). If none is given, a random order is chosen.
`seed`	(Optional) integer setting the state for random number generation. Defaults to seed = 123.
`nSlots`	(Optional) vector of length `nObjects` indicating the owners/slots possible/available for each object. (If nSlots only consists of ones this means that every object can only have one owner, if the numbers are higher interpreting objects for example as schools might be more intuitive). Defaults to a vector of length `nObjects` filled with ones.

Value

`rsd` returns a data frame containing the final matching of individuals (ind) to objects (obj).

Author(s)

Thilo Klein, Alexander Sauer

Examples

```
## Generate preference-matrix
prefs <- matrix(c(1,2,3,
                  3,1,2,
                  1,3,2),
                  byrow = FALSE, ncol = 3)

priority <- c(1,2,3)
nSlots <- c(1,1,1)

rsd(prefs = prefs, priority = priority, nSlots = nSlots)
```

`sri` *All stable matchings in the stable roommates problem with incomplete lists*

Description

Finds all stable matchings (if one exists) in the stable roommates problem with incomplete lists using the Prosser (2014) constraint encoding based on either given or randomly generated preferences.

Usage

```
sri(prefs = NULL, nAgents = NULL, seed = NULL, p.range = NULL)
```

Arguments

`prefs`	valuation matrix of dimension `nAgents x nAgents` that gives column-players' ranking over other players in decreasing order of preference (i.e. most preferred first).
`nAgents`	integer that gives the number of players in the market.
`seed`	integer setting the state for random number generation.
`p.range`	range of two intergers `p.range = c(p.min, p.max)`, where `p.min < p.max`. Produces incomplete preference lists with the length of each player's list randomly sampled from the range `[p.min, p.max]`. Note: interval is always too small because non-permissible matches are automatically deleted.

Value

`sri` returns a list with the following items.

`prefs`	agents' preference list.
`matching`	edgelist of matched pairs, inculding the number of the match (`matching`).

Author(s)

Thilo Klein

References

Gusfield, D.M. and R.W. Irving (1989). The Stable Marriage Problem: Structure and Algorithms, MIT Press.

Prosser, P. (2014). Stable Roommates and Constraint Programming. *Lecture Notes in Computer Science, CPAIOR 2014 Edition*. Springer International Publishing, 8451: 15–28.

Irving, R.W. and S. Scott (2007). The stable fixtures problem: A many-to-many extension of stable roommates. *Discrete Applied Mathematics*, 155: 2118–2129.

Examples

```
## Roommate problem with 10 players, given preferences:
 prefs <- matrix(rep(1:10, 10), 10, 10)
 sri(prefs=prefs)

## Roommate problem with 10 players, random preferences:
 sri(nAgents=10, seed=1)

## Roommate problem with no equilibrium matching:
 sri(nAgents=10, seed=2)

## Roommate problem with 3 equilibria:
 sri(nAgents=10, seed=3)
```

---

stabchk *Stability-Check*

Description

Checks a given two sided matching for blocking pairs.

Usage

```
stabchk(matching, c.prefs, s.prefs, nColleges = ncol(c.prefs),
  nStudents = ncol(s.prefs))
```

Arguments

`matching`	data frame or matrix of dimension (`min[nColleges, nStudents]`) x 2 containing in column 1 the colleges and in column 2 the students with each row forming a couple.
`c.prefs`	matrix of dimension `nStudents` x `nColleges` with column j containing college j'th ranking over students in decreasing order of preferences.
`s.prefs`	matrix of dimension `nColleges` x `nStudents` with column j containing student j'th ranking over colleges in decreasing order of preferences.
`nColleges`	integer indicating the number of colleges
`nStudents`	integer indicating the number of students

Value

stabchk returns a data frame with as many rows as blocking pairs were found. Column 1 indicates the college and column 2 indicate the student of the blocking pairs. Returns NULL if no blocking pair is found.

Author(s)

Thilo Klein, Alexander Sauer

Examples

```
## 1-a. Generate preferences for colleges
c.prefs = matrix(c(1,2,3,
                   3,2,1,
                   3,2,1),
                    byrow = FALSE, ncol = 3); c.prefs

## 1-b. Generate preferences for students
s.prefs = matrix(c(1,2,3,
                   3,2,1,
                   2,1,3),
                 byrow = FALSE, ncol = 3);s.prefs

## 1-c. Generate matching
matching = matrix(c(1,2,
                    2,1,
                    3,3),
                  byrow = TRUE, ncol = 2); matching

## 1-d. Check stability
stabchk(matching = matching, c.prefs = c.prefs, s.prefs = s.prefs)

## 2-a. Generate new matching without blocking pairs as a data frame
matching = data.frame('colleges' = c(1,2,3), 'student' = c(1,3,2))
stabchk(matching = matching, c.prefs = c.prefs, s.prefs = s.prefs)

## 3-a. Example with missing values:
matching  <- matrix(c(1,1,2,2,3,3), byrow = FALSE, ncol = 2)
c.prefs <- matrix(c(1,1,3,rep(NA, 6)), byrow = TRUE, ncol = 3)
s.prefs <- matrix(c(2,2,3,rep(NA, 6)), byrow = TRUE, ncol = 3)
stabchk(matching = matching, c.prefs = c.prefs, s.prefs = s.prefs)
```

---

stabit *Matching model and selection correction for group formation*

Description

The function provides a Gibbs sampler for a structural matching model that estimates preferences and corrects for sample selection bias when the selection process is a one-sided matching game; that is, group/coalition formation.

The input is individual-level data of all group members from one-sided matching marktes; that is, from group/coalition formation games.

In a first step, the function generates a model matrix with characteristics of *all feasible* groups of the same size as the observed groups in the market.

For example, in the stable roommates problem with $n = 4$ students $\{1, 2, 3, 4\}$ sorting into groups of 2, we have $\binom{4}{2} = 6$ feasible groups: (1,2)(3,4) (1,3)(2,4) (1,4)(2,3).

In the group formation problem with $n = 6$ students $\{1, 2, 3, 4, 5, 6\}$ sorting into groups of 3, we have $\binom{6}{3} = 20$ feasible groups. For the same students sorting into groups of sizes 2 and 4, we have $\binom{6}{2} + \binom{6}{4} = 30$ feasible groups.

The structural model consists of a selection and an outcome equation. The *Selection Equation* determines which matches are observed ($D = 1$) and which are not ($D = 0$).

$$\begin{aligned} D &= 1[V \in \Gamma] \\ V &= W\alpha + \eta \end{aligned}$$

Here, V is a vector of latent valuations of *all feasible* matches, ie observed and unobserved, and $1[.]$ is the Iverson bracket. A match is observed if its match valuation is in the set of valuations Γ that satisfy the equilibrium condition (see Klein, 2015a). This condition differs for matching games with transferable and non-transferable utility and can be specified using the `method` argument. The match valuation V is a linear function of W, a matrix of characteristics for *all feasible* groups, and η, a vector of random errors. α is a paramter vector to be estimated.

The *Outcome Equation* determines the outcome for *observed* matches. The dependent variable can either be continuous or binary, dependent on the value of the `binary` argument. In the binary case, the dependent variable R is determined by a threshold rule for the latent variable Y.

$$\begin{aligned} R &= 1[Y > c] \\ Y &= X\beta + \epsilon \end{aligned}$$

Here, Y is a linear function of X, a matrix of characteristics for *observed* matches, and ϵ, a vector of random errors. β is a paramter vector to be estimated.

The structural model imposes a linear relationship between the error terms of both equations as $\epsilon = \delta\eta + \xi$, where ξ is a vector of random errors and δ is the covariance paramter to be estimated. If δ were zero, the marginal distributions of ϵ and η would be independent and the selection problem would vanish. That is, the observed outcomes would be a random sample from the population of interest.

Usage

```
stabit(x, m.id = "m.id", g.id = "g.id", R = "R", selection = NULL,
  outcome = NULL, simulation = "none", seed = 123, max.combs = Inf,
  method = "NTU", binary = FALSE, offsetOut = 0, offsetSel = 0,
  marketFE = FALSE, censored = 0, gPrior = FALSE, dropOnes = FALSE,
  interOut = 0, interSel = 0, standardize = 0, niter = 10,
  verbose = FALSE)
```

Arguments

x	data frame with individual-level characteristics of all group members including market- and group-identifiers.
m.id	character string giving the name of the market identifier variable. Defaults to "m.id".
g.id	character string giving the name of the group identifier variable. Defaults to "g.id".
R	dependent variable in outcome equation. Defaults to "R".
selection	list containing variables and pertaining operators in the selection equation. The format is operation = "variable". See the Details and Examples sections.
outcome	list containing variables and pertaining operators in the outcome equation. The format is operation = "variable". See the Details and Examples sections.
simulation	should the values of dependent variables in selection and outcome equations be simulated? Options are "none" for no simulation, "NTU" for non-transferable utility matching, "TU" for transferable utility or "random" for random matching of individuals to groups. Simulation settings are (i) all model coefficients set to alpha=beta=1; (ii) covariance between error terms delta=0.5; (iii) error terms eta and xi are draws from a standard normal distribution.
seed	integer setting the state for random number generation if simulation=TRUE.
max.combs	integer (divisible by two) giving the maximum number of feasible groups to be used for generating group-level characteristics.
method	estimation method to be used. Either "NTU" or "TU" for selection correction using non-transferable or transferable utility matching as selection rule; "outcome" for estimation of the outcome equation only; or "model.frame" for no estimation.
binary	logical: if TRUE outcome variable is taken to be binary; if FALSE outcome variable is taken to be continuous.
offsetOut	vector of integers indicating the indices of columns in X for which coefficients should be forced to 1. Use 0 for none.
offsetSel	vector of integers indicating the indices of columns in W for which coefficients should be forced to 1. Use 0 for none.
marketFE	logical: if TRUE market-level fixed effects are used in outcome equation; if FALSE no market fixed effects are used.
censored	draws of the delta parameter that estimates the covariation between the error terms in selection and outcome equation are 0:not censored, 1:censored from below, 2:censored from above.
gPrior	logical: if TRUE the g-prior (Zellner, 1986) is used for the variance-covariance matrix.
dropOnes	logical: if TRUE one-group-markets are exluded from estimation.
interOut	two-colum matrix indicating the indices of columns in X that should be interacted in estimation. Use 0 for none.
interSel	two-colum matrix indicating the indices of columns in W that should be interacted in estimation. Use 0 for none.

standardize	numeric: if standardize>0 the independent variables will be standardized by dividing by standardize times their standard deviation. Defaults to no standardization standardize=0.
niter	number of iterations to use for the Gibbs sampler.
verbose	.

Details

Operators for variable transformations in selection and outcome arguments.

- add sum over all group members and divide by group size.
- int sum over all possible two-way interactions $x * y$ of group members and divide by the number of those, given by choose(n,2).
- ieq sum over all possible two-way equality assertions $1[x = y]$ and divide by the number of those.
- ive sum over all possible two-way interactions of vectors of variables of group members and divide by number of those.
- inv ...
- dst sum over all possible two-way distances between players and divide by number of those, where distance is defined as $e^{-|x-y|}$.

Author(s)

Thilo Klein

References

Klein, T. (2015a). Does Anti-Diversification Pay? A One-Sided Matching Model of Microcredit. *Cambridge Working Papers in Economics*, #1521.

Zellner, A. (1986). *On assessing prior distributions and Bayesian regression analysis with g-prior distributions*, volume 6, pages 233–243. North-Holland, Amsterdam.

Examples

```
## Not run:
## --- SIMULATED EXAMPLE ---

## 1. Simulate one-sided matching data for 200 markets (m=200) with 2 groups
##    per market (gpm=2) and 5 individuals per group (ind=5). True parameters
##    in selection equation is wst=1, in outcome equation wst=0.

## 1-a. Simulate individual-level, independent variables
 idata <- stabsim(m=200, ind=5, seed=123, gpm=2)
 head(idata)

## 1-b. Simulate group-level variables
 mdata <- stabit(x=idata, simulation="NTU", method="model.frame",
 selection = list(add="wst"), outcome = list(add="wst"), verbose=FALSE)
 head(mdata$OUT)
 head(mdata$SEL)
```

```
## 2. Bias from sorting

## 2-a. Naive OLS estimation
 lm(R ~ wst.add, data=mdata$OUT)$coefficients

## 2-b. epsilon is correlated with independent variables
 with(mdata$OUT, cor(epsilon, wst.add))

## 2-c. but xi is uncorrelated with independent variables
 with(mdata$OUT, cor(xi, wst.add))

## 3. Correction of sorting bias when valuations V are observed

## 3-a. 1st stage: obtain fitted value for eta
lm.sel <- lm(V ~ -1 + wst.add, data=mdata$SEL)
lm.sel$coefficients

eta <- lm.sel$resid[mdata$SEL$D==1]

## 3-b. 2nd stage: control for eta
 lm(R ~ wst.add + eta, data=mdata$OUT)$coefficients


## 4. Run Gibbs sampler
 fit1 <- stabit(x=idata, method="NTU", simulation="NTU", censored=1,
                selection = list(add="wst"), outcome = list(add="wst"),
                niter=2000, verbose=FALSE)


## 5. Coefficient table
 summary(fit1)


## 6. Plot MCMC draws for coefficients
 plot(fit1)


## --- REPLICATION, Klein (2015a) ---

## 1. Load data
 data(baac00); head(baac00)

## 2. Run Gibbs sampler
 klein15a <- stabit(x=baac00, selection = list(inv="pi",ieq="wst"),
        outcome = list(add="pi",inv="pi",ieq="wst",
        add=c("loan_size","loan_size2","lngroup_agei")), offsetOut=1,
        method="NTU", binary=TRUE, gPrior=TRUE, marketFE=TRUE, niter=800000)

## 3. Marginal effects
 summary(klein15a, mfx=TRUE)
```

```
## 4. Plot MCMC draws for coefficients
 plot(klein15a)

## End(Not run)
```

---

stabit2 *Matching model and selection correction for college admissions*

Description

The function provides a Gibbs sampler for a structural matching model that estimates preferences and corrects for sample selection bias when the selection process is a two-sided matching game; i.e., a matching of students to colleges.

The structural model consists of a selection and an outcome equation. The *Selection Equation* determines which matches are observed ($D = 1$) and which are not ($D = 0$).

$$\begin{aligned} D &= 1[V \in \Gamma] \\ V &= W\beta + \eta \end{aligned}$$

Here, V is a vector of latent valuations of *all feasible* matches, ie observed and unobserved, and $1[.]$ is the Iverson bracket. A match is observed if its match valuation is in the set of valuations Γ that satisfy the equilibrium condition (see Sorensen, 2007). The match valuation V is a linear function of W, a matrix of characteristics for *all feasible* matches, and η, a vector of random errors. β is a paramter vector to be estimated.

The *Outcome Equation* determines the outcome for *observed* matches. The dependent variable can either be continuous or binary, dependent on the value of the `binary` argument. In the binary case, the dependent variable R is determined by a threshold rule for the latent variable Y.

$$\begin{aligned} R &= 1[Y > c] \\ Y &= X\alpha + \epsilon \end{aligned}$$

Here, Y is a linear function of X, a matrix of characteristics for *observed* matches, and ϵ, a vector of random errors. α is a paramter vector to be estimated.

The structural model imposes a linear relationship between the error terms of both equations as $\epsilon = \kappa\eta + \nu$, where ν is a vector of random errors and κ is the covariance paramter to be estimated. If κ were zero, the marginal distributions of ϵ and η would be independent and the selection problem would vanish. That is, the observed outcomes would be a random sample from the population of interest.

Usage

```
stabit2(OUT = NULL, SEL = NULL, colleges = NULL, students = NULL,
  outcome = NULL, selection, binary = FALSE, niter, gPrior = FALSE,
  censored = 1, thin = 1, nCores = max(1, detectCores() - 1), ...)
```

Arguments

OUT	data frame with characteristics of all observed matches, including market identifier m.id, college identifier c.id and student identifier s.id.
SEL	optional: data frame with characteristics of all observed and unobserved matches, including market identifier m.id, college identifier c.id and student identifier s.id.
colleges	character vector of variable names for college characteristics. These variables carry the same value for any college.
students	character vector of variable names for student characteristics. These variables carry the same value for any student.
outcome	formula for match outcomes.
selection	formula for match valuations.
binary	logical: if TRUE outcome variable is taken to be binary; if FALSE outcome variable is taken to be continuous.
niter	number of iterations to use for the Gibbs sampler.
gPrior	logical: if TRUE the g-prior (Zellner, 1986) is used for the variance-covariance matrix. (Not yet implemented)
censored	draws of the kappa parameter that estimates the covariation between the error terms in selection and outcome equation are 0:not censored, 1:censored from below, 2:censored from above.
thin	integer indicating the level of thinning in the MCMC draws. The default thin=1 saves every draw, thin=2 every second, etc.
nCores	number of cores to be used in parallel Gibbs sampling.
...	.

Author(s)

Thilo Klein

References

Sorensen, M. (2007). How Smart is Smart Money? A Two-Sided Matching Model of Venture Capital. *Journal of Finance*, 62 (6): 2725-2762.

Examples

```
## Not run:
## --- SIMULATED EXAMPLE ---

## 1. Simulate two-sided matching data for 20 markets (m=20) with 100 students
##    (nStudents=100) per market and 20 colleges with quotas of 5 students, each
##    (nSlots=rep(5,20)). True parameters in selection and outcome equations are
##    all equal to 1.

xdata <- stabsim2(m=20, nStudents=100, nSlots=rep(5,20), verbose=FALSE,
  colleges = "c1", students = "s1",
```

```
  outcome = ~ c1:s1 + eta + nu,
  selection = ~ -1 + c1:s1 + eta
)
head(xdata$OUT)


## 2. Correction for sorting bias when match valuations V are observed

## 2-a. Bias from sorting
 lm1 <- lm(y ~ c1:s1, data=xdata$OUT)
 summary(lm1)

## 2-b. Cause of the bias
 with(xdata$OUT, cor(c1*s1, eta))

## 2-c. Correction for sorting bias
 lm2a <- lm(V ~ -1 + c1:s1, data=xdata$SEL); summary(lm2a)
 etahat <- lm2a$residuals[xdata$SEL$D==1]

 lm2b <- lm(y ~ c1:s1 + etahat, data=xdata$OUT)
 summary(lm2b)


## 3. Correction for sorting bias when match valuations V are unobserved

## 3-a. Run Gibbs sampler (when SEL is given)
 fit2 <- stabit2(OUT = xdata$OUT,
           SEL = xdata$SEL,
           outcome = y ~ c1:s1,
           selection = ~ -1 + c1:s1,
           niter=1000
 )

## 3-b. Alternatively: Run Gibbs sampler (when SEL is not given)
 fit2 <- stabit2(OUT = xdata$OUT,
           colleges = "c1",
           students = "s1",
           outcome = y ~ c1:s1,
           selection = ~ -1 + c1:s1,
           niter=1000
 )


## 4. Implemented methods

## 4-a. Get coefficients
 fit2

## 4-b. Coefficient table
 summary(fit2)

## 4-c. Get marginal effects
 summary(fit2, mfx=TRUE)
```

```
## 4-d. Also try the following functions
 #coef(fit2)
 #fitted(fit2)
 #residuals(fit2)
 #predict(fit2, newdata=NULL)


## 5. Plot MCMC draws for coefficients
 plot(fit2)

## End(Not run)
```

`stabsim` *Simulated data for group formation problem*

Description

Simulate individual-level data for one-sided matching markets.

Usage

```
stabsim(m, ind, seed = 123, singles = NULL, gpm = 2)
```

Arguments

`m`	integer indicating the number of markets to be simulated.
`ind`	integer (or vector) indicating the number of individuals per group.
`seed`	integer setting the state for random number generation. Defaults to `set.seed(123)`.
`singles`	integer giving the number of one-group markets.
`gpm`	integer giving the number of groups per market.

Value

`stabsim` returns a data frame with the randomly generated variables mimicking those in dataset `baac00`.

`m.id`	categorical: market identifier.
`g.id`	categorical: group identifier.
`wst`	binary: indicator taking the value 1 if last year was worse than the year before; 0 otherwise.
`R`	NA: group outcome is not simulated. It can be obtained using the `simulation` argument in function `stabit`.

Author(s)

Thilo Klein

Examples

```
## Coalitions [gpm := 2 !]
## Simulate one-sided matching data for 4 markets (m=4) with 2 groups
## per market (gpm=2) and 2 to 4 individuals per group (ind=2:4)
 idata <- stabsim(m=4, ind=2:4, seed=124, singles=2, gpm=2)

## Rommmates [ind := 2 !]
## Simulate one-sided matching data for 3 markets (m=3) with 3 groups
## per market (gpm=3) and 2 individuals per group (ind=2)
 idata <- stabsim(m=3, ind=2, seed=124, gpm=3)
```

---

stabsim2 *Simulated data for college admissions problem*

Description

Simulate data for two-sided matching markets. In the simulation for the Sorensen (2007) model with one selection equation, an equal sharing rule of $\lambda = 0.5$ is used.

Usage

```
stabsim2(m, nStudents, nColleges = length(nSlots), nSlots, colleges,
  students, outcome, selection, binary = FALSE, seed = 123,
  verbose = TRUE)
```

Arguments

m	integer indicating the number of markets to be simulated.
nStudents	integer indicating the number of students per market.
nColleges	integer indicating the number of colleges per market.
nSlots	vector of length nColleges indicating the number of places at each college, i.e. the college's quota.
colleges	character vector of variable names for college characteristics. These variables carry the same value for any college.
students	character vector of variable names for student characteristics. These variables carry the same value for any student.
outcome	formula for match outcomes.
selection	formula for match valuations.
binary	logical: if TRUE outcome variable is binary; if FALSE outcome variable is continuous.
seed	integer setting the state for random number generation. Defaults to set.seed(123).
verbose	.

Value

stabsim2 returns a list with the following items.

OUT

SEL

SELc

SELs

Author(s)

Thilo Klein

Examples

```
## Simulate two-sided matching data for 2 markets (m=2) with 10 students
## (nStudents=10) per market and 3 colleges (nColleges=3) with quotas of
## 2, 3, and 5 students, respectively.

xdata <- stabsim2(m=2, nStudents=10, nSlots=c(2,3,5), verbose=FALSE,
  colleges = "c1", students = "s1",
  outcome = ~ c1:s1 + eta + nu,
  selection = ~ -1 + c1:s1 + eta
)
head(xdata$OUT)
head(xdata$SEL)
```

---

ttc *Top-Trading-Cycles Algorithm with existing tenants*

Description

Implements an algorithm for the house allocation problem proposed by Abdulkadiroglu and Sonmez (1999) for a matching problem in which there are both vacant houses and existing tenants.

Usage

```
ttc(nStudents = ncol(s.prefs), nHouses = length(houses), s.prefs,
  houses, priority = NULL, seed = 123)
```

Arguments

nStudents	integer indicating the number of students. Defaults to ncol(s.prefs).
nHouses	integer indicating the number of houses. Defaults to length(houses).
s.prefs	matrix of dimension nHouses x nStudents with column j containing student jth ranking over houses in decreasing order of preferences (i.e. most preferred first).

houses	vector of length nHouses which represents the occupation of the houses. Entry in k contains j if student j is living in house k and NA if house k is vacant.
priority	(Optional) vector of length nStudents. Gives the prioirity ordering of the students, if nothing is specified a random ordering is chosen.
seed	(Optional) integer setting the state for random number generation. Defaults to seed =123.

Value

ttc returns a data frame of the matching of students (int) to houses (obj) for the house allocation problem based on the Top-Trading-Cycles algorithm.

Author(s)

Thilo Klein, Alexander Sauer

References

Abdulkadiroglu, A. and T. Sonmez (1999). House Allocation with Existing Tenants. *Journal of Economic Theory*, 88 (2): 233-260.

Shapley, L. and H. Scarf (1974). On Cores and Indivisibility. *Journal of Mathematical Economics*, 1(1): 23-37.

Examples

```
## Not run:
## 1-a. Generate matrix of individuals' preference rankings over objects,
## a.k.a. Rank Order Lists (ROL).
s.prefs <- matrix(c(3,2,4,1,        # ROL of student 1
                    3,5,6, NA,
                    3,1, NA,NA,
                    2,5,6,4,
                    1,3,2,NA,
                    2,4,5,6), nrow = 4, ncol = 6, byrow = FALSE); s.prefs

## 1-b. Generate vector of house occupation objects ('obj') and their owners ('ind')
(houses <- 1:6)

## 1-c. Find assignment based on TTC algorithm
ttc(s.prefs = s.prefs, houses = houses, nHouses = 6, priority = 1:6)

## 2-a.Compare the example in the paper Abdulkadiroglu et al. (1999)
## on page 246-248 (section 5.1 An Example):
## generate matrix of students' preference rankings over houses, a.k.a. Rank Order Lists (ROL)
s.prefs <- matrix(c(2,6,5,1,4,3,7,NA,
                  7,1,6,5,4,3,2,NA,
                  2,1,4,7,3,6,5,NA,
                  2,4,3,6,1,7,5,NA,
                  4,3,7,1,2,5,6,NA), byrow = FALSE, ncol= 5); s.prefs

## 2-b. Generate house occupation, so student 1 lives in house 1, ..., student 4 lives in house 4
```

```
## and the other houses are vacant.
houses <- c(1,2,3,4,NA,NA,NA,NA); houses

## 2-c. Generate priority ordering
priority <- 1:5

## 2-d. Find assigment
ttc(s.prefs = s.prefs, houses = houses, priority = priority)

## End(Not run)
```

---

ttc2 *Top-Trading-Cycles Algorithm for a two sided matching problem*

Description

Implements the school matching algorithm proposed in Abdulkadiroglu and Sonmez (2003) for a matching problem in which both sides have preferences. Missing preferences are handled in the following ways: Suppose that a student only ranked colleges that are already matched to other students. This student is removed from the matching process and a list with all unmatchable students is printed. If `full_return` is set to `TRUE`, a vector with this students is returned as well. Now suppose during the matching process a student points to a college that still has capacities but does not rank any more students. We assume now that the college is indifferent over all other students (so we do not allow for free capacieties) and we match the student who wants to go there to the college.

Usage

```
ttc2(nStudents = ncol(s.prefs), nColleges = ncol(c.prefs),
  s.prefs = NULL, c.prefs = NULL, nSlots = NULL, priority = NULL,
  seed = 123, full_return = FALSE)
```

Arguments

`nStudents`	integer indicating the number of students in the matching problem. Defaults to `ncol(s.prefs)`.
`nColleges`	integer indicating the number of colleges in the matching problem. Defaults to `ncol(c.prefs)`.
`s.prefs`	matrix of dimension `nColleges` x `nStudents` with the jth column containing student j's ranking over colleges in decreasing order of preference (i.e. most preferred first).
`c.prefs`	matrix of dimension `nStudents` x `nColleges` with the ith column containing college i's ranking over students in decreasing order of preference (i.e. most preferred first).
`nSlots`	vector of length `nColleges` indicating the number of places (i.e. quota) of each college.

priority	(Optional) vector of length nStudents. Gives the prioirity ordering of the students, if nothing is specified a random ordering is chosen.
seed	(Optional) integer setting the state for random number generation. Defaults to seed =123.
full_return	(Optinal) If TRUE the return value is a list with the matching, the remaining seats and the unmatchable students is returned. Defaults to FALSE and only the matching is returned.

Value

ttc2 returns a data frame of the matching of students (ind) to colleges (obj) for the school market problem based on the Top-Trading-Cycles algorithm.

Author(s)

Thilo Klein, Alexander Sauer

References

Abdulkadiroglu, A. and T. Sonmez (2003). School Choice: A Mechanism Design Approach. *American Economic Review*, 93 (3): 729-747.

Examples

```
## Not run:
## 1-a. Compare example from the Abdulkadiroglu et al. (2003) (in the Appendix, page 742-744)
## 1-b. Generate matrix of students' preference rankings over schools, a.k.a. Rank Order Lists (ROL)
s.prefs <- matrix(c(
                  2,1,3,4,
                  1,2,3,4,
                  3,2,1,4,
                  3,4,1,2,
                  1,3,4,2,
                  4,1,2,3,
                  1,2,3,4,
                  1,2,4,3),
                  byrow = FALSE, ncol = 8); s.prefs

## 1-c. Generate matrix of schools' preference rankings over students, a.k.a. Rank Order Lists (ROL)
c.prefs <- matrix(c(
                  1,2,3,4,5,6,7,8,
                  3,5,4,8,7,2,1,6,
                  5,3,1,7,2,8,6,4,
                  6,8,7,4,2,3,5,1),
                  byrow = FALSE, ncol = 4); c.prefs

## 1-d. Generate capacities
nSlots <- c(2,2,3,3)

## 1-e. Find assignment based on TTC algorithm
ttc2(s.prefs = s.prefs, c.prefs = c.prefs, nSlots = nSlots)
```

```
## 2-a. Generate college preferences with college 1 only ranking student 1
c.prefs <- matrix(c(
                   1,rep(NA,7),
                   3,5,4,8,7,2,1,6,
                   5,3,1,7,2,8,6,4,
                   6,8,7,4,2,3,5,1),
                   byrow = FALSE, ncol = 4); c.prefs

## 2-b. Find assignment based on TTC algorithm
ttc2(s.prefs = s.prefs, c.prefs = c.prefs, nSlots = nSlots, priority = 1:8)

## If all schools have the same preferences the two sided ttc and the serial dictator yield
## the same outcome if the preferences are taken to be the prioirty order for the serial dictator

# Preferences are the same for all schools:
c.prefs <- matrix(c(
                  5,3,1,7,2,8,6,4,
                  5,3,1,7,2,8,6,4,
                  5,3,1,7,2,8,6,4,
                  5,3,1,7,2,8,6,4),
                  byrow = FALSE, ncol = 4)
priority <- c.prefs[,1]

match_ttc <- ttc2(s.prefs = s.prefs, c.prefs = c.prefs, nSlots = nSlots); match_ttc
match_sd <- rsd(prefs = s.prefs, priority = priority, nSlots = nSlots); match_sd
all(match_ttc == match_sd)

## End(Not run)
```

`ttcc` *Top-Trading-Cycles and Chains Algorithm*

Description

Implements the Top Trading Cycle and Chains algorithm proposed by Roth et al. (2004) for the kidney exchange problem. The algorithm requires a rule to determine which chain will be used if there is more than one possibility. The chosen rule is to search for the longest chain and remove it from the problem (even the first kidney which was unassigned).

Usage

```
ttcc(nPatients = ncol(prefs), prefs, priority = NULL, seed = 123)
```

Arguments

`nPatients`	integer indicating the number of patient/donor-pairs in the matching problem. Defaults to `ncol(prefs)`.

prefs	matrix of dimension (nPatients + 1) x nPatients with column j containg patients jth ranking over kidneys in decreasing order of preferences (i.e. most preferred first). An entry with value (nPatients +1) indicates that the patient prefers the waiting list to all kidney below in his ranking (therefore they do not matter and can be neglected/NA).
priority	(Optional) Vector of length nPatients. Gives the prioirty ordering of the patient, if nothing is specified a random ordering is chosen.
seed	(Optional) integer setting the state for random number generation. Defaults to seed = 123.

Value

ttcc returns a list with the matching and a vector containing the patients who are assigned to the waiting list. The matching comprises a data frame of the operations to be performed between patient-donor pairs (ind-obj).

Author(s)

Thilo Klein, Alexander Sauer

References

Roth, A.; T. Sonmez; U. Unver (2004). Kidney Exchange. *Quarterly Journal of Economics*, 119 (2): 457-488.

Examples

```
## Not run:
## Compare Example 1 from Roth et al. (2004) on page 469 - 475
## generate matrix of patients' preference rankings over kidneys, a.k.a. Rank Order Lists (ROL)

prefs <- matrix(c( 9,10, 1,NA,NA,NA,NA,
                  11, 3, 5, 6, 2,NA,NA,
                   2, 4, 5, 6, 7, 8,13,
                   5, 9, 1, 8,10, 3,13,
                   3, 7,11, 4, 5,NA,NA,
                   3, 5, 8, 6,NA,NA,NA,
                   6, 1, 3, 9,10, 1,13,
                   6, 4,11, 2, 3, 8,NA,
                   3,11,13,NA,NA,NA,NA,
                  11, 1, 4, 5, 6, 7,13,
                   3, 6, 5,11,NA,NA,NA,
                  11, 3, 9, 8,10,12,NA),
             byrow = FALSE, ncol = 12); prefs
priority <- 1:12
ttcc(prefs = prefs, priority = priority)
## The final matching differs slightly because in Round 3 another chain is chosen due to a different
## decision rule (compare Figure 3, p472. Here W1 instead of W2 is chosen)

## End(Not run)
```

Index