Package 'matchingR'

January 26, 2018

Type Package

Title Matching Algorithms in R and C++

Version 1.3.0

Date 2018-01-26

Author Jan Tilly, Nick Janetos

Maintainer Jan Tilly <jantilly@gmail.com>

Description Computes matching algorithms quickly using Rcpp. Implements the Gale-Shapley Algorithm to compute the stable matching for two-sided markets, such as the stable marriage problem and the college-admissions problem. Implements Irving's Algorithm for the stable roommate problem. Implements the top trading cycle algorithm for the indivisible goods trading problem.

License GPL (>= 2)

URL https://github.com/jtilly/matchingR/

BugReports https://github.com/jtilly/matchingR/issues/

Depends Rcpp

LinkingTo Rcpp, RcppArmadillo

Suggests testthat, knitr

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-01-26 22:04:41 UTC

R topics documented:

`matchingR-package` *matchingR: Matching Algorithms in R and C++*

Description

matchingR is an R package which quickly computes a variety of matching algorithms for one-sided and two-sided markets. This package implements

- the Gale-Shapley Algorithm to compute the stable matching for two-sided markets, such as the stable marriage problem and the college-admissions problem
- Irving's Algorithm to compute the stable matching for one-sided markets such as the stable roommates problem
- the top trading cycle algorithm for the indivisible goods trading problem.

All matching algorithms are implemented in `C++` and can therefore be computed quickly. The package may be useful when the number of market participants is large or when many matchings need to be computed (e.g. for extensive simulations or for estimation purposes). The Gale-Shapley function of this package has successfully been used to simulate preferences and compute the matching with 30,000 participants on each side of the market.

Matching markets are common in practice and widely studied by economists. Popular examples include

- the National Resident Matching Program that matches graduates from medical school to residency programs at teaching hospitals throughout the United States

- the matching of students to schools including the New York City High School Match or the the Boston Public School Match (and many more)
- the matching of kidney donors to recipients in kidney exchanges.

Author(s)

Jan Tilly, Nick Janetos

References

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

Irving, R. W. (1985). An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6(4): 577–595

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

See Also

Useful links:

- https://github.com/jtilly/matchingR/
- Report bugs at https://github.com/jtilly/matchingR/issues/

Examples

```
# stable marriage problem
set.seed(1)
nmen = 25
nwomen = 20
uM = matrix(runif(nmen*nwomen), nrow=nwomen, ncol=nmen)
uW = matrix(runif(nwomen*nmen), nrow=nmen, ncol=nwomen)
results = galeShapley.marriageMarket(uM, uW)
galeShapley.checkStability(uM, uW, results$proposals, results$engagements)

# college admissions problem
nstudents = 25
ncolleges = 5
uStudents = matrix(runif(nstudents*ncolleges), nrow=ncolleges, ncol=nstudents)
uColleges = matrix(runif(nstudents*ncolleges), nrow=nstudents, ncol=ncolleges)
results = galeShapley.collegeAdmissions(studentUtils = uStudents,
                                        collegeUtils = uColleges,
                                        slots = 4)
results
# check stability
galeShapley.checkStability(uStudents,
                           uColleges,
                           results$matched.students,
                           results$matched.colleges)

# stable roommate problem
```

```
set.seed(2)
N = 10
u = matrix(runif(N^2),  nrow = N, ncol = N)
results = roommate(utils = u)
results
# check stability
roommate.checkStability(utils = u, matching = results)

# top trading cycle algorithm
N = 10
u = matrix(runif(N^2),  nrow = N, ncol = N)
results = toptrading(utils = u)
results
# check stability
toptrading.checkStability(utils = u, matching = results)
```

`cpp_wrapper_galeshapley` *C++ wrapper for Gale-Shapley Algorithm*

Description

This function provides an R wrapper for the C++ backend. Users should not call this function directly and instead use `galeShapley.marriageMarket` or `galeShapley.collegeAdmissions`.

Usage

```
cpp_wrapper_galeshapley(proposerPref, reviewerUtils)
```

Arguments

`proposerPref`	is a matrix with the preference order of the proposing side of the market. If there are `n` proposers and `m` reviewers in the market, then this matrix will be of dimension `m` by `n`. The `i,j`th element refers to `j`'s `i`th most favorite partner. Preference orders must be complete and specified using C++ indexing (starting at 0).
`reviewerUtils`	is a matrix with cardinal utilities of the courted side of the market. If there are `n` proposers and `m` reviewers, then this matrix will be of dimension `n` by `m`. The `i,j`th element refers to the payoff that individual `j` receives from being matched to individual `i`.

Value

A list with elements that specify who is matched to whom. Suppose there are `n` proposers and `m` reviewers. The list contains the following items:

- `proposals` is a vector of length `n` whose `i`th element contains the number of the reviewer that proposer `i` is matched to using C++ indexing. Proposers that remain unmatched will be listed as being matched to `m`.

- engagements is a vector of length m whose jth element contains the number of the proposer that reviewer j is matched to using C++ indexing. Reviwers that remain unmatched will be listed as being matched to n.

---

cpp_wrapper_galeshapley_check_stability

C++ Wrapper to Check Stability of Two-sided Matching

Description

This function checks if a given matching is stable for a particular set of preferences. This function provides an R wrapper for the C++ backend. Users should not call this function directly and instead use galeShapley.checkStability.

Usage

```
cpp_wrapper_galeshapley_check_stability(proposerUtils, reviewerUtils, proposals,
  engagements)
```

Arguments

proposerUtils	is a matrix with cardinal utilities of the proposing side of the market. If there are n proposers and m reviewers, then this matrix will be of dimension m by n. The i, jth element refers to the payoff that individual j receives from being matched to individual i.
reviewerUtils	is a matrix with cardinal utilities of the courted side of the market. If there are n proposers and m reviewers, then this matrix will be of dimension n by m. The i, jth element refers to the payoff that individual j receives from being matched to individual i.
proposals	is a matrix that contains the number of the reviewer that a given proposer is matched to: the first row contains the number of the reviewer that is matched with the first proposer (using C++ indexing), the second row contains the id of the reviewer that is matched with the second proposer, etc. The column dimension accommodates proposers with multiple slots.
engagements	is a matrix that contains the number of the proposer that a given reviewer is matched to (using C++ indexing). The column dimension accommodates reviewers with multiple slots.

Value

true if the matching is stable, false otherwise

cpp_wrapper_irving *Computes a stable roommate matching*

Description

This is the C++ wrapper for the stable roommate problem. Users should not call this function directly, but instead use roommate.

Usage

```
cpp_wrapper_irving(pref)
```

Arguments

pref	is a matrix with the preference order of each individual in the market. If there are n individuals, then this matrix will be of dimension n-1 by n. The i, jth element refers to j's ith most favorite partner. Preference orders must be specified using C++ indexing (starting at 0). The matrix pref must be of dimension n-1 by n.

Value

A vector of length n corresponding to the matchings that were formed (using C++ indexing). E.g. if the 4th element of this vector is 0 then individual 4 was matched with individual 1. If no stable matching exists, then this function returns a vector of zeros. @export

cpp_wrapper_irving_check_stability *Check if a matching solves the stable roommate problem*

Description

This function checks if a given matching is stable for a particular set of preferences. This function checks if there's an unmatched pair that would rather be matched with each other than with their assigned partners.

Usage

```
cpp_wrapper_irving_check_stability(pref, matchings)
```

Arguments

pref	is a matrix with the preference order of each individual in the market. If there are n individuals, then this matrix will be of dimension n-1 by n. The i, jth element refers to j's ith most favorite partner. Preference orders must be specified using C++ indexing (starting at 0). The matrix pref must be of dimension n-1 by n.
matchings	is a vector of length n corresponding to the matchings that were formed (using C++ indexing). E.g. if the 4th element of this vector is 0 then individual 4 was matched with individual 1. If no stable matching exists, then this function returns a vector of zeros.

Value

true if the matching is stable, false otherwise @export

---

cpp_wrapper_ttc *Computes the top trading cycle algorithm*

Description

This is the C++ wrapper for the top trading cycle algorithm. Users should not call this function directly, but instead use toptrading.

Usage

```
cpp_wrapper_ttc(pref)
```

Arguments

pref	is a matrix with the preference order of all individuals in the market. If there are n individuals, then this matrix will be of dimension n by n. The i, jth element refers to j's ith most favorite partner. Preference orders must be specified using C++ indexing (starting at 0).

Details

This function uses the top trading cycle algorithm to find a stable trade between agents, each with some indivisible good, and with preferences over the goods of other agents. Each agent is matched to one other agent, and matchings are not necessarily two-way. Agents may be matched with themselves.

Value

A vector of length n corresponding to the matchings being made, so that e.g. if the 4th element is 5 then agent 4 was matched to agent 6. This vector uses C++ indexing that starts at 0.

cpp_wrapper_ttc_check_stability

Check if a one-sided matching for the top trading cycle algorithm is stable

Description

Check if a one-sided matching for the top trading cycle algorithm is stable

Usage

```
cpp_wrapper_ttc_check_stability(pref, matchings)
```

Arguments

pref	is a matrix with the preference order of all individuals in the market. If there are n individuals, then this matrix will be of dimension n by n. The i,jth element refers to j's ith most favorite partner. Preference orders must be specified using C++ indexing (starting at 0).
matchings	is a vector of length n corresponding to the matchings being made, so that e.g. if the 4th element is 5 then agent 4 was matched to agent 6. This vector uses C++ indexing that starts at 0.

Value

true if the matching is stable, false otherwise

galeShapley.checkPreferences

Check if preference order is complete

Description

This function checks if a given preference ordering is complete. If needed, it transforms the indices from R indices (starting at 1) to C++ indices (starting at zero).

Usage

```
galeShapley.checkPreferences(pref)
```

Arguments

pref	is a matrix with ordinal preference orderings for one side of the market. Suppose that pref refers to the preferences of n women over m men. In that case, pref will be of dimension m by n. The i,jth element refers to woman j's ith most favorite man. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).

Value

a matrix with ordinal preference orderings with proper C++ indices or NULL if the preference order is not complete.

Examples

```
# preferences in proper C++ indexing: galeShapley.checkPreferences(pref)
# will return pref
pref = matrix(c(0, 1, 0,
                1, 0, 1), nrow = 2, ncol = 3, byrow = TRUE)
pref
galeShapley.checkPreferences(pref)

# preferences in R indexing: galeShapley.checkPreferences(pref)
# will return pref-1
pref = matrix(c(1, 2, 1,
                2, 1, 2), nrow = 2, ncol = 3, byrow = TRUE)
pref
galeShapley.checkPreferences(pref)

# incomplete preferences: galeShapley.checkPreferences(pref)
# will return NULL
pref = matrix(c(3, 2, 1,
                2, 1, 2), nrow = 2, ncol = 3, byrow = TRUE)
pref
galeShapley.checkPreferences(pref)
```

---

galeShapley.checkStability

Check if a two-sided matching is stable

Description

This function checks if a given matching is stable for a particular set of preferences. This stability check can be applied to both the stable marriage problem and the college admission problem. The function requires preferences to be specified in cardinal form. If necessary, the function `rankIndex` can be used to turn ordinal preferences into cardinal utilities.

Usage

```
galeShapley.checkStability(proposerUtils, reviewerUtils, proposals, engagements)
```

Arguments

`proposerUtils`	is a matrix with cardinal utilities of the proposing side of the market. If there are `n` proposers and `m` reviewers, then this matrix will be of dimension `m` by `n`. The `i,j`th element refers to the payoff that proposer `j` receives from being matched to reviewer `i`.

`reviewerUtils` is a matrix with cardinal utilities of the courted side of the market. If there are `n` proposers and `m` reviewers, then this matrix will be of dimension `n` by `m`. The `i,j`th element refers to the payoff that reviewer `j` receives from being matched to proposer `i`.

`proposals` is a matrix that contains the number of the reviewer that a given proposer is matched to: the first row contains the reviewer that is matched to the first proposer, the second row contains the reviewer that is matched to the second proposer, etc. The column dimension accommodates proposers with multiple slots.

`engagements` is a matrix that contains the number of the proposer that a given reviewer is matched to. The column dimension accommodates reviewers with multiple slots.

Value

true if the matching is stable, false otherwise

Examples

```
# define cardinal utilities
uM = matrix(c(0.52, 0.85,
              0.96, 0.63,
              0.82, 0.08,
              0.55, 0.34), nrow = 4, byrow = TRUE)
uW = matrix(c(0.76, 0.88, 0.74, 0.02,
              0.32, 0.21, 0.02, 0.79), ncol = 4, byrow = TRUE)
# define matching
results = list(
     proposals = matrix(c(2, 1), ncol = 1),
     engagements = matrix(c(2, 1, NA, NA), ncol = 1))
# check stability
galeShapley.checkStability(uM, uW, results$proposals, results$engagements)

# if preferences are in ordinal form, we can use galeShapley.validate
# to transform them into cardinal form and then use checkStability()
prefM = matrix(c(2, 1,
                 3, 2,
                 4, 4,
                 1, 3), nrow = 4, byrow = TRUE)
prefW = matrix(c(1, 1, 1, 2,
                 2, 2, 2, 1), ncol = 4, byrow = TRUE)
# define matching
results = list(proposals = matrix(c(2, 1), ncol = 1),
               engagements = matrix(c(2, 1, NA, NA), ncol = 1))
# check stability
pref.validated = galeShapley.validate(proposerPref = prefM,
                                      reviewerPref = prefW)
galeShapley.checkStability(pref.validated$proposerUtils,
                           pref.validated$reviewerUtils,
                           results$proposals,
                           results$engagements)
```

`galeShapley.collegeAdmissions` *Gale-Shapley Algorithm: College Admissions Problem*

Description

This function computes the Gale-Shapley algorithm and finds a solution to the college admissions problem. In the student-optimal college admissions problem, `n` students apply to `m` colleges, where each college has `s` slots.

Usage

```
galeShapley.collegeAdmissions(studentUtils = NULL, collegeUtils = NULL,
  studentPref = NULL, collegePref = NULL, slots = 1,
  studentOptimal = TRUE)
```

Arguments

`studentUtils`	is a matrix with cardinal utilities of the students. If there are `n` students and `m` colleges, then this matrix will be of dimension `m` by `n`. The `i,j`th element refers to the payoff that student `j` receives from being matched to college `i`.
`collegeUtils`	is a matrix with cardinal utilities of colleges. If there are `n` students and `m` colleges, then this matrix will be of dimension `n` by `m`. The `i,j`th element refers to the payoff that college `j` receives from being matched to student `i`.
`studentPref`	is a matrix with the preference order of the proposing side of the market (only required when `studentUtils` is not provided). If there are `n` students and `m` colleges in the market, then this matrix will be of dimension `m` by `n`. The `i,j`th element refers to student `j`'s `i`th most favorite college. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).
`collegePref`	is a matrix with the preference order of the courted side of the market (only required when `collegeUtils` is not provided). If there are `n` students and `m` colleges in the market, then this matrix will be of dimension `n` by `m`. The `i,j`th element refers to individual `j`'s `i`th most favorite partner. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).
`slots`	is the number of slots that each college has available. If this is 1, then the algorithm is identical to `galeShapley.marriageMarket`. `slots` can either be a integer or a vector. If it is an integer, then all colleges have the same number of slots. If it is a vector, it must have as many elements as there are colleges where each element refers to the number of slots at a particular college.
`studentOptimal`	is `TRUE` if students apply to colleges. The resulting match is student-optimal. `studentOptimal` is `FALSE` if colleges apply to students. The resulting match is college-optimal.

Details

The algorithm works analogously to galeShapley.marriageMarket. The Gale-Shapley algorithm works as follows: Students ("the proposers") sequentially make proposals to each of their most preferred available colleges ("the reviewers"). A college can hold on to at most `s` proposals at a time. A college with an open slot will accept any application that it receives. A college that already holds on to `s` applications will reject any application by a student that it values less than her current set of applicants. If a college receives an application from a student that it values more than its current set of applicants, then it will accept the application and drop its least preferred current applicant. This process continues until all students are matched to colleges.

The Gale-Shapley Algorithm requires a complete specification of students' and colleges' preferences over each other. Preferences can be passed on to the algorithm in ordinal form (e.g. student 3 prefers college 1 over college 3 over college 2) or in cardinal form (e.g. student 3 receives payoff 3.14 from being matched to college 1, payoff 2.51 from being matched to college 3 and payoff 2.13 from being matched to college 2). Preferences must be complete, i.e. all students must have fully specified preferences over all colleges and vice versa.

In the version of the algorithm that is implemented here, all individuals – colleges and students – prefer being matched to anyone to not being matched at all.

The algorithm still works with an unequal number of students and slots. In that case some students will remain unmatched or some slots will remain open.

Value

A list with elements that specify which student is matched to which college and who remains unmatched. Suppose there are `n` students and `m` colleges with `s` slots. The list contains the following items:

- `matched.students` is a vector of length `n` whose `i`th element contains college that student `i` is matched to. Students that remain unmatched will be listed as being matched to college `NA`.
- `matched.colleges` is a matrix of dimension `m` by `s` whose `j`th row contains the students that were admitted to college `j`. Slots that remain open show up as being matched to student to `NA`.
- `unmatched.students` is a vector that lists the remaining unmatched students This vector will be empty when all students get matched.
- `unmatched.colleges` is a vector that lists colleges with open slots. If a college has multiple open slots, it will show up multiple times. This vector will be empty whenever all college slots get filled.

Examples

```
ncolleges = 10
nstudents = 25

# randomly generate cardinal preferences of colleges and students
collegeUtils = matrix(runif(ncolleges*nstudents), nrow=nstudents, ncol=ncolleges)
studentUtils = matrix(runif(ncolleges*nstudents), nrow=ncolleges, ncol=nstudents)

# run the student-optimal algorithm
results.studentoptimal = galeShapley.collegeAdmissions(studentUtils = studentUtils,
                                collegeUtils = collegeUtils,
```

```
                              slots = 2,
                              studentOptimal = TRUE)
results.studentoptimal

# run the college-optimal algorithm
results.collegeoptimal = galeShapley.collegeAdmissions(studentUtils = studentUtils,
                              collegeUtils = collegeUtils,
                              slots = 2,
                              studentOptimal = FALSE)
results.collegeoptimal

# transform the cardinal utilities into preference orders
collegePref = sortIndex(collegeUtils)
studentPref = sortIndex(studentUtils)

# run the student-optimal algorithm
results.studentoptimal = galeShapley.collegeAdmissions(studentPref = studentPref,
                             collegePref = collegePref,
                             slots = 2,
                             studentOptimal = TRUE)
results.studentoptimal

# run the college-optimal algorithm
results.collegeoptimal = galeShapley.collegeAdmissions(studentPref = studentPref,
                             collegePref = collegePref,
                             slots = 2,
                             studentOptimal = FALSE)
results.collegeoptimal
```

`galeShapley.marriageMarket` *Gale-Shapley Algorithm: Stable Marriage Problem*

Description

This function computes the Gale-Shapley algorithm and finds a solution to the stable marriage problem.

Usage

```
galeShapley.marriageMarket(proposerUtils = NULL, reviewerUtils = NULL,
  proposerPref = NULL, reviewerPref = NULL)
```

Arguments

`proposerUtils`	is a matrix with cardinal utilities of the proposing side of the market. If there are `n` proposers and `m` reviewers, then this matrix will be of dimension `m` by `n`. The `i,j`th element refers to the payoff that proposer `j` receives from being matched to proposer `i`.

`reviewerUtils` is a matrix with cardinal utilities of the courted side of the market. If there are `n` proposers and `m` reviewers, then this matrix will be of dimension `n` by `m`. The `i,j`th element refers to the payoff that reviewer `j` receives from being matched to proposer `i`.

`proposerPref` is a matrix with the preference order of the proposing side of the market. This argument is only required when `proposerUtils` is not provided. If there are `n` proposers and `m` reviewers in the market, then this matrix will be of dimension `m` by `n`. The `i,j`th element refers to proposer `j`'s `i`th most favorite reviewer. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).

`reviewerPref` is a matrix with the preference order of the courted side of the market. This argument is only required when `reviewerUtils` is not provided. If there are `n` proposers and `m` reviewers in the market, then this matrix will be of dimension `n` by `m`. The `i,j`th element refers to reviewer `j`'s `i`th most favorite proposer. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).

Details

The Gale-Shapley algorithm works as follows: Single men ("the proposers") sequentially make proposals to each of their most preferred available women ("the reviewers"). A woman can hold on to at most one proposal at a time. A single woman will accept any proposal that is made to her. A woman that already holds on to a proposal will reject any proposal by a man that she values less than her current match. If a woman receives a proposal from a man that she values more than her current match, then she will accept the proposal and her previous match will join the line of bachelors. This process continues until all men are matched to women.

The Gale-Shapley Algorithm requires a complete specification of proposers' and reviewers' preferences over each other. Preferences can be passed on to the algorithm in ordinal form (e.g. man 3 prefers woman 1 over woman 3 over woman 2) or in cardinal form (e.g. man 3 receives payoff 3.14 from being matched to woman 1, payoff 2.51 from being matched to woman 3, and payoff 2.15 from being matched to woman 2). Preferences must be complete, i.e. all proposers must have fully specified preferences over all reviewers and vice versa.

In the version of the algorithm that is implemented here, all individuals – proposers and reviewers – prefer being matched to anyone to not being matched at all.

The algorithm still works with an unequal number of proposers and reviewers. In that case some agents will remain unmatched.

This function can also be called using `galeShapley`.

Value

A list with elements that specify who is matched to whom and who remains unmatched. Suppose there are `n` proposers and `m` reviewers. The list contains the following items:

- `proposals` is a vector of length `n` whose `i`th element contains the number of the reviewer that proposer `i` is matched to. Proposers that remain unmatched will be listed as being matched to `NA`.

- `engagements` is a vector of length `m` whose `j`th element contains the number of the proposer that reviewer `j` is matched to. Reviwers that remain unmatched will be listed as being matched to `NA`.
- `single.proposers` is a vector that lists the remaining single proposers. This vector will be empty whenever `n<=m`.
- `single.reviewers` is a vector that lists the remaining single reviewers. This vector will be empty whenever `m<=n`.

See Also

`galeShapley.collegeAdmissions`

Examples

```
nmen = 5
nwomen = 4
# generate cardinal utilities
uM = matrix(runif(nmen*nwomen), nrow = nwomen, ncol = nmen)
uW = matrix(runif(nwomen*nmen), nrow = nmen, ncol = nwomen)
# run the algorithm using cardinal utilities as inputs
results = galeShapley.marriageMarket(uM, uW)
results

# transform the cardinal utilities into preference orders
prefM = sortIndex(uM)
prefW = sortIndex(uW)
# run the algorithm using preference orders as inputs
results = galeShapley.marriageMarket(proposerPref = prefM, reviewerPref = prefW)
results
```

`galeShapley.validate` *Input validation of preferences*

Description

This function parses and validates the arguments that are passed on to the Gale-Shapley Algorithm. In particular, it checks if user-defined preference orders are complete and returns an error otherwise. If user-defined orderings are given in terms of R indices (starting at 1), then these are transformed into C++ indices (starting at zero).

Usage

```
galeShapley.validate(proposerUtils = NULL, reviewerUtils = NULL,
  proposerPref = NULL, reviewerPref = NULL)
```

Arguments

proposerUtils	is a matrix with cardinal utilities of the proposing side of the market. If there are n proposers and m reviewers, then this matrix will be of dimension m by n. The i,jth element refers to the payoff that proposer j receives from being matched to reviewer i.
reviewerUtils	is a matrix with cardinal utilities of the courted side of the market. If there are n proposers and m reviewers, then this matrix will be of dimension n by m. The i,jth element refers to the payoff that reviewer j receives from being matched to proposer i.
proposerPref	is a matrix with the preference order of the proposing side of the market (only required when proposerUtils is not provided). If there are n proposers and m reviewers in the market, then this matrix will be of dimension m by n. The i,jth element refers to proposer j's ith most favorite reviewer. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).
reviewerPref	is a matrix with the preference order of the courted side of the market (only required when reviewerUtils is not provided). If there are n proposers and m reviewers in the market, then this matrix will be of dimension n by m. The i,jth element refers to reviewer j's ith most favorite proposer. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).

Value

a list containing proposerUtils, reviewerUtils, proposerPref (reviewerPref are not required after they are translated into reviewerUtils).

Examples

```
# market size
nmen = 5
nwomen = 4

# generate cardinal utilities
uM = matrix(runif(nmen*nwomen), nrow = nwomen, ncol = nmen)
uW = matrix(runif(nwomen*nmen), nrow = nmen, ncol = nwomen)

# turn cardinal utilities into ordinal preferences
prefM = sortIndex(uM)
prefW = sortIndex(uW)

# validate cardinal preferences
preferences = galeShapley.validate(uM, uW)
preferences

# validate ordinal preferences
preferences = galeShapley.validate(proposerPref = prefM, reviewerPref = prefW)
preferences
```

```
# validate ordinal preferences when these are in R style indexing
# (instead of C++ style indexing)
preferences = galeShapley.validate(proposerPref = prefM + 1, reviewerPref = prefW + 1)
preferences

# validate preferences when proposer-side is cardinal and reviewer-side is ordinal
preferences = galeShapley.validate(proposerUtils = uM, reviewerPref = prefW)
preferences
```

`matchingR-deprecated` *Deprecated Functions in matchingR*

Description

These functions are provided for compatibility with older version of the matchingR package. Eventually, these functions will be removed.

Usage

```
validateInputs(...)
```

Arguments

`...`	generic set of parameters — see documentation of new functions

Details

validateInputs	was replaced by galeShapley.validate
checkStability	was replaced by galeShapley.checkStability
checkPreferenceOrder	was replaced by galeShapley.checkPreferences
one2many	now mapped into galeShapley.collegeAdmissions
many2one	now mapped into galeShapley.collegeAdmissions
one2one	was replaced by galeShapley.marriageMarket
galeShapleyMatching	was replaced by cpp_wrapper_galeshapley
stableRoommateMatching	was replaced by cpp_wrapper_irving
onesided	was replaced by roommate
checkStabilityRoommate	was replaced by cpp_wrapper_irving_check_stability
validateInputsOneSided	was replaced by roommate.validate
checkPreferenceOrderOnesided	was replaced by roommate.checkPreferences
topTradingCycle	was replaced by cpp_wrapper_ttc
checkStabilityTopTradingCycle	was replaced by cpp_wrapper_ttc_check_stability

rankIndex *Rank elements within column of a matrix*

Description

This function returns the rank of each element within each column of a matrix. The highest element receives the highest rank.

Usage

```
rankIndex(sortedIdx)
```

Arguments

sortedIdx	is the input matrix

Value

a rank matrix

repcol *Repeat each column of a matrix n times*

Description

This function repeats each column of a matrix n times

Usage

```
repcol(x, n)
```

Arguments

x	is the input matrix
n	is the number of repetitions (can be a vector)

Value

matrix with repeated columns

reprow *Repeat each row of a matrix n times*

Description

This function repeats each row of a matrix n times

Usage

```
reprow(x, n)
```

Arguments

x	is the input matrix
n	is the number of repetitions (can be a vector)

Value

matrix with repeated rows

roommate *Compute matching for one-sided markets*

Description

This function computes the Irving (1985) algorithm for finding a stable matching in a one-sided matching market.

Usage

```
roommate(utils = NULL, pref = NULL)
```

Arguments

utils	is a matrix with cardinal utilities for each individual in the market. If there are n individuals, then this matrix will be of dimension n-1 by n. Column j refers to the payoff that individual j receives from being matched to individual 1, 2,..., j-1, j+1, ...n. If a square matrix is passed as utils, then the main diagonal will be removed.
pref	is a matrix with the preference order of each individual in the market. This argument is only required when utils is not provided. If there are n individuals, then this matrix will be of dimension n-1 by n. The i,jth element refers to j's ith most favorite partner. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0). The matrix pref must be of dimension n-1 by n. Otherwise, the function will throw an error.

Details

Consider the following example: A set of n potential roommates, each with ranked preferences over all the other potential roommates, are to be matched to rooms, two roommates per room. A matching is stable if there is no roommate r1 that would rather be matched to some other roommate d2 than to his current roommate r2 and the other roommate d2 would rather be matched to r1 than to his current roommate d1.

The algorithm works in two stages. In the first stage, all participants begin unmatched, then, in sequence, begin making proposals to other potential roommates, beginning with their most preferred roommate. If a roommate receives a proposal, he either accepts it if he has no other proposal which is better, or rejects it otherwise. If this stage ends with a roommate who has no proposals, then there is no stable matching and the algorithm terminates.

In the second stage, the algorithm proceeds by finding and eliminating rotations. Roughly speaking, a rotation is a sequence of pairs of agents, such that the first agent in each pair is least preferred by the second agent in that pair (of all the agents remaining to be matched), the second agent in each pair is most preferred by the first agent in each pair (of all the agents remaining to be matched) and the second agent in the successive pair is the second most preferred agent (of the agents remaining to be matched) of the first agent in the succeeding pair, where here 'successive' is taken to mean 'modulo m', where m is the length of the rotation. Once a rotation has been identified, it can be eliminated in the following way: For each pair, the second agent in the pair rejects the first agent in the pair (recall that the second agent hates the first agent, while the first agent loves the second agent), and the first agent then proceeds to propose to the second agent in the succeeding pair. If at any point during this process, an agent no longer has any agents left to propose to or be proposed to from, then there is no stable matching and the algorithm terminates.

Otherwise, at the end, every agent is left proposing to an agent who is also proposing back to them, which results in a stable matching.

Note that neither existence nor uniqueness is guaranteed, this algorithm finds one matching, not all of them. If no matching exists, this function returns NULL.

Value

A vector of length n corresponding to the matchings that were formed. E.g. if the 4th element of this vector is 6 then individual 4 was matched with individual 6. If no stable matching exists, then this function returns NULL.

Examples

```
# example using cardinal utilities
utils = matrix(c(-1.63, 0.69, -1.38, -0.03,
                 2.91, -0.52, 0.52, 0.22,
                 0.53, -0.52, -1.18, 0.53), byrow=TRUE, ncol = 4, nrow = 3)
utils
results = roommate(utils = utils)
results

# example using preference orders
pref = matrix(c(3, 1, 2, 3,
                4, 3, 4, 2,
                2, 4, 1, 1), byrow = TRUE, ncol = 4)
```

```
pref
results = roommate(pref = pref)
results
```

`roommate.checkPreferences` *Check if preference order for a one-sided market is complete*

Description

Check if preference order for a one-sided market is complete

Usage

```
roommate.checkPreferences(pref)
```

Arguments

`pref`	is a matrix with the preference order of each individual in the market. This argument is only required when `utils` is not provided. If there are `n` individuals, then this matrix will be of dimension `n-1` by `n`. The `i,j`th element refers to `j`'s `i`th most favorite partner. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0). The matrix `pref` must be of dimension `n-1` by `n`. Otherwise, the function will throw an error.

Value

a matrix with preference orderings with proper C++ indices or NULL if the preference order is not complete.

`roommate.checkStability` *Check if a roommate matching is stable*

Description

This function checks if a particular roommate matching is stable. A matching is stable if there is no roommate `r1` that would rather be matched to some other roommate `d2` than to his current roommate `r2` and the other roommate `d2` would rather be matched to `r1` than to his current roommate `d1`.

Usage

```
roommate.checkStability(utils = NULL, pref = NULL, matching)
```

Arguments

utils	is a matrix with cardinal utilities for each individual in the market. If there are n individuals, then this matrix will be of dimension n-1 by n. Column j refers to the payoff that individual j receives from being matched to individual 1, 2,..., j-1, j+1, ...n. If a square matrix is passed as utils, then the main diagonal will be removed.
pref	is a matrix with the preference order of each individual in the market. This argument is only required when utils is not provided. If there are n individuals, then this matrix will be of dimension n-1 by n. The i,jth element refers to j's ith most favorite partner. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0). The matrix pref must be of dimension n-1 by n. Otherwise, the function will throw an error.
matching	is a vector of length n corresponding to the matchings that were formed. E.g. if the 4th element of this vector is 6 then individual 4 was matched with individual 6.

Value

true if stable, false if not

Examples

```
# define preferences
pref = matrix(c(3, 1, 2, 3,
                4, 3, 4, 2,
                2, 4, 1, 1), byrow = TRUE, ncol = 4)
pref
# compute matching
results = roommate(pref = pref)
results
# check if matching is stable
roommate.checkStability(pref = pref, matching = results)
```

---

roommate.validate *Input validation for one-sided markets*

Description

This function parses and validates the arguments for the function roommate. It returns the validates arguments. This function is called as part of roommate. Only one of the arguments needs to be provided.

Usage

```
roommate.validate(utils = NULL, pref = NULL)
```

Arguments

utils	is a matrix with cardinal utilities for each individual in the market. If there are n individuals, then this matrix will be of dimension n-1 by n. Column j refers to the payoff that individual j receives from being matched to individual 1, 2,..., j-1, j+1, ...n. If a square matrix is passed as utils, then the main diagonal will be removed.
pref	is a matrix with the preference order of each individual in the market. This argument is only required when utils is not provided. If there are n individuals, then this matrix will be of dimension n-1 by n. The i,jth element refers to j's ith most favorite partner. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0). The matrix pref must be of dimension n-1 by n. Otherwise, the function will throw an error.

Value

The validated preference ordering using C++ indexing.

---

sortIndex *Sort indices of a matrix within a column*

Description

Within each column of a matrix, this function returns the indices of each element in descending order

Usage

```
sortIndex(u)
```

Arguments

u	is the input matrix with cardinal preferences

Value

a matrix with sorted indices (the agents' ordinal preferences)

sortIndexOneSided *Ranks elements with column of a matrix, assuming a one-sided market.*

Description

Returns the rank of each element with each column of a matrix. So, if row 34 is the highest number for column 3, then the first row of column 3 will be 34 – unless it is column 34, in which case it will be 35, to adjust for the fact that this is a single-sided market.

Usage

```
sortIndexOneSided(u)
```

Arguments

u	A matrix with agents' cardinal preferences. Column i is agent i's preferences.

Value

a matrix with the agents' ordinal preferences

toptrading *Compute the top trading cycle algorithm*

Description

This package implements the top trading cycle algorithm.

Usage

```
toptrading(utils = NULL, pref = NULL)
```

Arguments

utils	is a matrix with cardinal utilities of all individuals in the market. If there are n individuals, then this matrix will be of dimension n by n. The i,jth element refers to the payoff that individual j receives from being matched to individual i.
pref	is a matrix with the preference order of all individuals in the market. This argument is only required when utils is not provided. If there are n individuals, then this matrix will be of dimension n by n. The i,jth element refers to j's ith most favorite partner. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).

Details

The top trading algorithm solves the following problem: A set of n agents each currently own their own home, and have preferences over the homes of other agents. The agents may trade their homes in some way, the problem is to identify a set of trades between agents so that no subset of agents can defect from the rest of the group, and by trading within themselves improve their own payoffs.

Roughly speaking, the top trading cycle proceeds by identifying cycles of agents, then eliminating those cycles until no agents remain. A cycle is a sequence of agents such that each agent most prefers the next agent's home (out of the remaining unmatched agents), and the last agent in the sequence most prefers the first agent in the sequence's home.

The top trading cycle is guaranteed to produce a unique outcome, and that outcome is the unique outcome in the core, meaning there is no other outcome with the stability property described above.

Value

A vector of length n corresponding to the matchings being made, so that e.g. if the 4th element is 6 then agent 4 was matched to agent 6.

Examples

```
# example using cardinal utilities
utils = matrix(c(-1.4, -0.66, -0.45, 0.03,
                 0.72, 1.71, 0.59, 0.07,
                 0.44, 1.76, 1.71, -0.27,
                 0.26, 2.18, 1.4, 0.12), byrow = TRUE, nrow = 4)
utils
results = toptrading(utils = utils)
results

# example using ordinal preferences
pref = matrix(c(2, 4, 3, 4,
                3, 3, 4, 2,
                4, 2, 2, 1,
                1, 1, 1, 3), byrow = TRUE, nrow = 4)
pref
results = toptrading(pref = pref)
results
```

---

toptrading.checkStability

Check if there are any pairs of agents who would rather swap houses with each other rather than be with their own two current respective partners.

Description

Check if there are any pairs of agents who would rather swap houses with each other rather than be with their own two current respective partners.

Usage

```
toptrading.checkStability(utils = NULL, pref = NULL, matchings)
```

Arguments

utils	is a matrix with cardinal utilities of all individuals in the market. If there are n individuals, then this matrix will be of dimension n by n. The i, jth element refers to the payoff that individual j receives from being matched to individual i.
pref	is a matrix with the preference order of all individuals in the market. This argument is only required when utils is not provided. If there are n individuals, then this matrix will be of dimension n by n. The i, jth element refers to j's ith most favorite partner. Preference orders can either be specified using R-indexing (starting at 1) or C++ indexing (starting at 0).
matchings	is a vector of length n corresponding to the matchings being made, so that e.g. if the 4th element is 6 then agent 4 was matched to agent 6.

Value

true if the matching is stable, false otherwise

Examples

```
pref = matrix(c(2, 4, 3, 4,
                3, 3, 4, 2,
                4, 2, 2, 1,
                1, 1, 1, 3), byrow = TRUE, nrow = 4)
pref
results = toptrading(pref = pref)
results
toptrading.checkStability(pref = pref, matchings = results)
```

Index