R documentation

of all in 'man/'

August 27, 2015

R topics documented:

SDG-package 1
avgGen 2
barComparison 6
compare 7
dinGen 8
extendQ 12
irtGen 13
minGen 16
oimage 19
poksGen 20
poksMatrixGen 23
QgenInc 24
QgenReg 25
randGen 26
reduceQ 28
skillsGen 29
successProbGen 30

Index **32**

`SDG-package` *Synthetic Data Generation for the Educational Data Mining. SDG*

Description

The goal of this package is to generate synthetic data. The package is very flexible while introducing parameters and very adaptive depending on the information the user owns. The structure of the library is based on a collection of functions that generate students' performance data according to different models. Then, there are also functions to generate different types of matrices and functions to plot the information in a really useful and understandable way.

Details

Package:	SDG
Type:	Package
Version:	1.0
Date:	2015-02-17

Author(s)

Javier Asenjo Villamayor
Supervisor: Michel C. Desmarais

Maintainer: Who to complain to <javier.asenjo.villamayor@gmail.com>

References

- Beheshti, B. and Desmarais, M.C. (2014). Assessing Model Fit With Synthetic vs. Real Data, Polytechnique Montreal.

- Beheshti, B. and Desmarais, M.C. (2014). Predictive performance of prevailing approaches to skills assessment techniques: Insights from real vs. synthetic data sets 7th Conference on Educational Data Data Mining (EDM 2014), London, England, p. 409-410.

-Desmarais, M.C. and Pelczer, I. (2010). On the Faithfullness of Simulated Student Performance Data. In Proceedings of Educational Data Mining 2010 (EDM2010). Pittsburg, PA, Jun. 11-13, p. 21-30.

See Also

CDM - package : http://cran.r-project.org/web/packages/CDM/
Partitions - package : http://cran.r-project.org/web/packages/partitions/index.html

Examples

```
library(SDG)
```

`avgGen` *Performance data generation according to an average approach. Linear model.*

Description

This function generates students' performance data according to a Q-matrix sampling model where more than one skill can be required to succeed an item.
Given a set of skills involved in the set of items, there is a matrix where each element represents the probability of a student to suceed an item which involves the correspondent skill. It's a students per skills matrix.[0-1] values.
According to the information provided by the Q-Matrix, if an item requires 2 skills, the probability of a student to answer correctly to that item will be the root mean square of the probabilities of this student to succeed items involving both skills.

Usage

```
avgGen(its, sts, rank, q = NULL, skills = NULL, mean = 0, deviation = 1)
```

Arguments

its	Number of items.
sts	Number of students.
rank	Number of skills the set of items has.
q	Q-Matrix representing the skills required to succeed an item. skills per item matrix.0,1 values. NULL by default.
skills	Matrix representing the probability of each student to answer correctly to an item involving a specific skill. students per skills matrix.[0-1] values. NULL by default.
mean	If the skills matrix is not provided to the function, this parameter is the mean used to generate that matrix according to the standard distribution. 0 by default.
deviation	If the skills matrix is not provided to the function, this parameter is the standard deviation used to generate that matrix according to the standard distribution. 1 by default.

Value

A list with the following information:

results	Students per items matrix where each element represents if the student answered correctly (1) or not (0) to the correspondent item.
skills.matrix	A student per skill matrix where each element represents the probability of a student to answer correctly to an item that involves the correspondent skill.
q.matrix	The Q-matrix used.

See Also

QgenInc for the external generation of a Q-matrix.
QgenReg for the external generation of a Q-matrix.
extendQ for the external generation of a Q-matrix.
reduceQ for the external generation of a Q-matrix.
skillsGen for the external generation of the skills matrix.

Examples

```
#####EXAMPLE 1 : Generation introducing the minimum number of parameters required #####

 # We generate student performance data about 3 students answering to 5 items and 3 skills.

 result <- avgGen(its = 5, sts = 3, rank = 3)

 # We extract the correspondent information

 performance  <- result$results
 performance
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    0    0    1
```

```
[2,]    1    0    0    0    0
[3,]    1    0    1    1    1

skills  <- result$skills.matrix
skills
        [,1]      [,2]      [,3]
[1,] 0.5659460 0.7730670 0.1510989
[2,] 0.2754485 0.3557638 0.5745654
[3,] 0.0414596 0.1636330 0.9992421

q.matrix  <- result$q.matrix
q.matrix
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    0    1    0
[2,]    1    1    0    1    1
[3,]    1    0    1    1    1

#####EXAMPLE 2 : Generation modeling the mean and the standard deviation used #####

# We generate student performance data about 3 students answering to 5 items and 3 skills.
For the stantard distribution (skills matrix), we use a mean of 0.5 and a standard
deviation of 0.2.

result <- minGen(its = 5, sts = 3, rank = 3,mean=0.5,deviation=0.2)

# We extract the correspondent information

performance  <- result$results
performance
      [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    1    1
[2,]    1    0    0    0    1
[3,]    1    0    1    1    1

skills  <- result$skills.matrix
skills
        [,1]      [,2]      [,3]
[1,] 0.6575190 0.6271332 0.7383802
[2,] 0.7441959 0.5290334 0.6381248
[3,] 0.6988804 0.5926346 0.6011667

q.matrix  <- result$q.matrix
q.matrix
      [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    1
[2,]    1    1    0    1    0
[3,]    1    1    1    1    0

#####EXAMPLE 3 : Generation introducing a Q-matrix #####

#We generate a Q-Matrix with 3 skills involved.

q.matrix <- QgenInc(num.skills = 3,maxSkillsPerItem=2)
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    1    1    0
[2,]    0    1    0    1    0    1
```

```
[3,]    0    0    1    0    1    1

# We generate student performance data about 3 students answering to 6 items, 3 skills
and the Q Matrix we want. For the stantard distribution (skills matrix), we use a mean
of 0.5 and a standard deviation of 0.2.

 result <- minGen(its = 6, sts = 3, rank = 3,q = q.matrix,mean=0.5,deviation=0.2)

 # We extract the correspondent information

 performance  <- result$results
 performance
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    1    1    0    1    0    1
 [2,]    0    0    0    0    1    0
 [3,]    1    0    1    0    0    0

 skills  <- result$skills.matrix
 skills
        [,1]      [,2]      [,3]
 [1,] 0.7293017 0.6951063 0.6243320
 [2,] 0.8131855 0.6306849 0.7204342
 [3,] 0.6953339 0.6937963 0.5450721

 q.matrix  <- result$q.matrix
 q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    1    0    0    1    1    0
 [2,]    0    1    0    1    0    1
 [3,]    0    0    1    0    1    1

 #####EXAMPLE 4 : Generation introducing a Q-Matrix and a skills matrix #####

 #We generate a Q-Matrix with 3 skills involved.

 q.matrix <- QgenInc(num.skills = 3,maxSkillsPerItem=2)
 q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    1    0    0    1    1    0
 [2,]    0    1    0    1    0    1
 [3,]    0    0    1    0    1    1

 #We generate a skills matrix  skills for 3 students, 3 skills, a mean of 0 and a
 standard deviation of 1.

 skills  <- skillsGen(sts = 3,mean = 0,deviation = 1,rank = 3)
 skills
        [,1]      [,2]      [,3]
 [1,] 0.003903748 0.9802114 0.5885409
 [2,] 0.900522227 0.3580975 0.4001344
 [3,] 0.978809077 0.9618280 0.1626563

 # We generate student performance data about 3 students answering to 6 items, 3
 skills and the Q Matrixand skills matrix we want.

 result <- minGen(its = 6, sts = 3, rank = 3,q = q.matrix,skills = skills)
```

```
# We extract the correspondent information

performance  <- result$results
performance
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    1    1    0    0    1
[2,]    1    1    0    0    0    0
[3,]    1    1    1    1    0    0

skills  <- result$skills.matrix
skills
        [,1]      [,2]      [,3]
[1,] 0.003903748 0.9802114 0.5885409
[2,] 0.900522227 0.3580975 0.4001344
[3,] 0.978809077 0.9618280 0.1626563

q.matrix  <- result$q.matrix
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    1    1    0
[2,]    0    1    0    1    0    1
[3,]    0    0    1    0    1    1
```

---

barComparison *Bar graph*

Description

This function generates a bar graph to compare the performance of different approaches.

Usage

```
barComparison(results, labels, col)
```

Arguments

results	Vector with the values to be compared.
labels	Vector with the names of the values to be plotted.
col	Vector of colors for the bars.

Examples

```
x <- randGen(its = 5, sts = 3)$result
 x
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    1    0    1
 [2,]    0    1    1    1    0
 [3,]    0    1    0    1    1

 y <- randGen(its = 5, sts = 3)$result
 y
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    0    1    1    1
```

```
[2,]    1    0    0    1    1
[3,]    1    0    0    0    0

z <- randGen(its = 5, sts = 3)$result
z
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    0    0    1
[2,]    1    0    0    0    0
[3,]    0    1    1    1    1

result1  <- compare (x,y)
result2  <- compare(x,z)
results  <- c(result1,result2)
labels <- c('y','z')
col <- c('red','green')
barComparison(results,labels,col)
```

`compare` *Percentage of similarity between two {0,1} matrices.*

Description

This function returns the performance reached by a {0,1} matrix, comparing it to a reference (expected) matrix.

Usage

```
compare(reference, result)
```

Arguments

`reference`	Matrix used as reference.
`result`	Matrix to analyze its performance or its similarity to the reference matrix.

Value

The percentage of similarity comparing element by element

Examples

```
x <- randGen(its = 5, sts = 3)$result
 x
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    1    0    1
 [2,]    0    1    1    1    0
 [3,]    0    1    0    1    1

 y <- randGen(its = 5, sts = 3)$result
 y
      [,1] [,2] [,3] [,4] [,5]
 [1,]    1    0    1    1    1
 [2,]    1    0    0    0    0
 [3,]    0    0    1    0    1
```

```
similarity <- compare(reference = x, result = y)
similarity
[1] 80
```

dinGen *Generation according to the DINA/DINO model.*

Description

This function generates the students' performance data according to a Q-matrix sampling model where more than one skill can be required to succeed an item.
Given a set of skills involved in the set of items, there is a matrix where each element represents the probability of a student mastering the correspondent skill. It's a students per skills matrix. {0,1} values.
According to the information provided by the Q-Matrix, if an item requires 2 skills, the probability of a student to answer correctly to that item will depend on the model chosen and the skills the student masters.
If the DINA model is chosen and the student masters both skills, he will answer wrong with a probability corresponding to the slip value proper to this item. Otherwise, if he doesn't master both skills, he will get a chance to answer right corresponding to the guess parameter.
If the DINO model is chosen and the student masters one of both skills, he will answer wrong with a probability corresponding to the slip value proper to this item. Otherwise, if he doesn't master any of the skills involved, he will get a chance to answer right corresponding to the guess parameter.

Usage

```
dinGen(its, sts, rank, q = NULL, guess = rep(0.2, nrow(q)), slip = guess,
  skills = NULL, mean = 0, deviation = 1, model = "DINA")
```

Arguments

`its`	Number of items.
`sts`	Number of students.
`rank`	Number of skills the set of items has.
`q`	Q-Matrix representing the skills required to succeed an item. Skills per item matrix.0,1 values. NULL by default.
`guess`	A vector with the guess parameter for each item. If the student doesn't master the skills required to succeed the correspondent item, the probability of succeeding that item will be the guess value corresponding to this item. 0.2 by default.
`slip`	A vector with the slip parameter for each item. If the student master the skills required to succeed the correspondent item, the probability of answering wrong to that item will be the slip value corresponding to this item. Same as the guess parameter by default.
`skills`	Matrix representing the probability of each student to answer correctly to an item involving a specific skill. Students per skills matrix.This matrix will help to generate the skills mastery matrix. A probability above 0.5 will be considered as mastering the skill.[0-1] values. NULL by default.
`mean`	If the skills matrix is not provided to the function, this parameter is the mean used to generate that matrix according to the standard distribution. 0 by default.

`deviation` If the skills matrix is not provided to the function, this parameter is the standard deviation used to generate that matrix according to the standard distribution. 1 by default.

`model` Model chosen between 'DINA' and 'DINO'. 'DINA' by default.

Value

A list with the following information:

`results` Students per items matrix where each element represents if the student answered correctly (1) or not (0) to the correspondent item.

`skills.mastery` A student per skill matrix where each element represents if a student masters a skill or not.

`q.matrix` The Q-matrix used.

References

http://cran.r-project.org/web/packages/CDM/

See Also

`QgenInc` for the external generation of a Q-matrix.
`QgenReg` for the external generation of a Q-matrix.
`extendQ` for the external generation of a Q-matrix.
`reduceQ` for the external generation of a Q-matrix.
`skillsGen` for the external generation of the skills matrix.

Examples

```
#####EXAMPLE 1 : Generation introducing the minimum number of parameters required (DINA model) #####

 # We generate student performance data about 3 students answering to 5 items and 3 skills. DINA model.

 result <- dinGen(its = 5,sts = 3,rank = 3)

 # We extract the correspondent information

 performance <- result$results
 performance
       [,1] [,2] [,3] [,4] [,5]
 [1,]    1    1    0    0    1
 [2,]    0    0    1    0    1
 [3,]    0    0    1    1    0

 skills.mastery  <- result$skills.mastery
 skills.mastery
       [,1] [,2] [,3]
 [1,]    1    1    0
 [2,]    1    0    1
 [3,]    0    1    1

 q.matrix  <- result$q.matrix
 q.matrix
       [,1] [,2] [,3] [,4] [,5]
```

```
[1,]    0    1    0    0    1
[2,]    1    1    0    0    0
[3,]    0    0    1    1    1


#####EXAMPLE 2 : Generation modeling the skills matrix parameters (DINO model)#####

# We generate student performance data about 3 students answering to 5 items and 3
 skills. DINO model. For the stantard distribution (skills matrix), we use a mean
 of 0.5 and a standard deviation of 0.2.

result <- dinGen(its = 5,sts = 3,rank = 3,mean = 0.5,deviation = 0.2,model = 'DINO')

# We extract the correspondent information

performance <- result$results
performance
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    1    1    0
[2,]    1    1    0    1    1
[3,]    1    0    1    1    1

skills.mastery  <- result$skills.mastery
skills.mastery
    [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1

q.matrix  <- result$q.matrix
q.matrix
      [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    0
[2,]    0    0    0    1    0
[3,]    1    1    0    0    1

#####EXAMPLE 3 : Generation introducing a Q-Matrix (DINA model)#####

#We generate a Q-Matrix with 3 skills involved.

q.matrix <- QgenInc(num.skills = 3, maxSkillsPerItem = 3)
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    0    0    1    1    0    1
[2,]    0    1    0    1    0    1    1
[3,]    0    0    1    0    1    1    1

# We generate student performance data about 3 students answering to 7 items and
3 skills. DINA model.

result <- dinGen(its = 7,sts = 3,rank = 3,q = q.matrix)

# We extract the correspondent information

performance <- result$results
performance
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
```

```
[1,]    1    0    0    0    0    0    0
[2,]    0    0    1    0    0    0    1
[3,]    1    1    1    1    0    1    1

skills.mastery  <- result$skills.mastery
skills.mastery
      [,1] [,2] [,3]
[1,]    1    1    0
[2,]    0    1    0
[3,]    1    1    1

q.matrix  <- result$q.matrix
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    0    0    1    1    0    1
[2,]    0    1    0    1    0    1    1
[3,]    0    0    1    0    1    1    1


#####EXAMPLE 4 : Generation introducing a Q-Matrix and a skills mastery matrix
(DINO model)#####

#We generate a Q-Matrix with 3 skills involved.

q.matrix <- QgenInc(num.skills = 3, maxSkillsPerItem = 3)
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    0    0    1    1    0    1
[2,]    0    1    0    1    0    1    1
[3,]    0    0    1    0    1    1    1
#We generate a skills matrix  skills for 3 students, 3 skills, a mean of 0 and
a standard deviation of 1.

skills  <- skillsGen(sts = 3,mean = 0,deviation = 1,rank = 3)
skills
        [,1]      [,2]      [,3]
[1,] 0.8362343 0.5610519 0.9333690
[2,] 0.7390838 0.8986089 0.6403717
[3,] 0.3655802 0.7970677 0.3658878

# We generate student performance according to the Q-matrix and de skills matrix generated.
DINO model.

result <- dinGen(its = 7,sts = 3,rank = 3,q = q.matrix,skills=skills,model='DINO')

# We extract the correspondent information

performance <- result$results
performance
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    0    1    0    0    1    0    1
[2,]    1    1    1    1    1    0    0
[3,]    0    1    0    0    0    1    1

skills.mastery  <- result$skills.mastery
skills.mastery
      [,1] [,2] [,3]
```

```
[1,]    1    1    1
[2,]    1    1    1
[3,]    0    1    0

q.matrix  <- result$q.matrix
q.matrix
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    0    0    1    1    0    1
[2,]    0    1    0    1    0    1    1
[3,]    0    0    1    0    1    1    1
```

---

extendQ *Q-matrix extension.*

Description

This function increases the number of items of a Q-Matrix randomly repeating some of its columns. It adds the number of items (columns) introduced as a parameter.

Usage

```
extendQ(q, extraItems)
```

Arguments

q	Q-matrix to extend.
extraItems	Number of items(columns) to add to the Q-Matrix

Value

q	Q matrix with the information about the skills required to succeed each item. Skills per items matrix.

See Also

QgenReg for the generation of a Q-Matrix .
QgenInc for the generation of a Q-Matrix .
reduceQ to modify the size of the a Q-matrix.

Examples

```
#We generate a Q-matrix with 4 skills involved and 3 skills as maximum involved per item.

 q.matrix <- QgenInc(num.skills = 4, maxSkillsPerItem = 3)
 q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
 [1,]    1    0    0    0    1    1    1    0    0     0     1     1     1     0
 [2,]    0    1    0    0    1    0    0    1    1     0     1     1     0     1
 [3,]    0    0    1    0    0    1    0    1    0     1     1     0     1     1
 [4,]    0    0    0    1    0    0    1    0    1     1     0     1     1     1

 # We add 2 more items
```

```
new.qmatrix <- extendQ(q = q.matrix, extraItems = 2)
new.qmatrix
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
[1,]    1    0    0    0    1    1    1    0    0     0     1     1     1     0     1     1
[2,]    0    1    0    0    1    0    0    1    1     0     1     1     0     1     0     1
[3,]    0    0    1    0    0    1    0    1    0     1     1     0     1     1     0     0
[4,]    0    0    0    1    0    0    1    0    1     1     0     1     1     1     0     1
```

`irtGen` *Generation according to the IRT model.*

Description

This function generates students' performance data according to the IRT model. There is no Q-matrix or skills matrix involved.
Each item has a difficulty 'b' and a discrimination parameter 'a'. IRT assumes that the probability of success to an item X_j is a function of a single ability factor θ :

$$P(X_j = 1|\theta) = (1/(1 + e^(- a_j * (\theta - b_j))))$$

The ability of each student is generated using the normal distribution with parameters that can be modeled. It's normalized between -4 and 4.
The items' difficulties will be generated according to the normal distribution with modifiables parameters as well. It's normalized between -3 and 3.
However, a students' performance matrix,representing the success or failure of a set of students to a set of items, can be provided so as the abilities and item difficulty matrices can be generated learning from that given matrix.

Moreover, there is also the possibility of introducing a slip and guess noise. In that case an expected success (1) could become a failure (0) depending on the slip parameter. In the same way, an expected failure (0) could become a success (1) depending on the guess parameter.

Usage

```
irtGen(its, sts, result = NULL, meanAb = 0, sdAb = 1.5, maxAb = 2.75,
  minAb = -2.75, meanDifIt = 0, sdDifIt = 1, maxItDif = 2.5,
  minItDif = -2.5, meanDisc = 1, sdDisc = 0, guess = 0.2, slip = 0.2)
```

Arguments

`its`	Number of items.
`sts`	Number of students.
`result`	Students' performance matrix where each element represents if a student i has answered correctly (1) or not (0) to an item j. NULL by default.
`meanAb`	If the results matrix is not provided. The abilities matrix will be generated according to the standard distribution with this parameter as the mean. 0 by default.
`sdAb`	If the results matrix is not provided. The abilities matrix will be generated according to the standard distribution with this parameter as the standard deviation. Since the abilities matrix is a [-4,4] matrix, this parameter takes the value 2.5 by default.

`maxAb`	If the results matrix is not provided. The abilities matrix will be generated according to the standard distribution.with this parameter as the mean. This parameter is the maximum ability allowed for a student .2.5 by default.
`minAb`	If the results matrix is not provided. The abilities matrix will be generated according to the standard distribution.with this parameter as the mean. This parameter is the minimum ability allowed for a student .-2.5 by default.
`meanDisc`	The items' discrimination matrix will be generated according to the normal distribution with the value of this parameter as the mean. 0 by default.
`sdDisc`	The items' discrimination matrix will be generated according to the normal distribution with the value of this parameter as the standard deviation. 1 by default.
`guess`	Even though according to the IRT approach the student has answered correctly to an item. The user can introduce a slip parameter.
`slip`	Even though according to the IRT approach the student hasn't answered correctly to an item. The user can introduce a guess parameter.
`meanDiffIt`	If the results matrix is not provided. The items' difficulty matrix will be generated according to the standard distribution with this parameter as the mean. 0 by default.
`sdDiffIt`	If the results matrix is not provided. The items' difficulty matrix will be generated according to the standard distribution with this parameter as the standard deviation. Since the abilities matrix is a [-3,3] matrix, this parameter takes the value 1 by default.
`maxDiffIt`	If the results matrix is not provided. The abilities matrix will be generated according to the standard distribution with this parameter as the mean. This parameter is the maximum item difficulty allowed. 2.5 by default.
`minDiffIt`	If the results matrix is not provided. The abilities matrix will be generated according to the standard distribution with this parameter as the mean. This parameter is the minimum item difficulty allowed. 2.5 by default.

Value

A list with the following information:

`results`	Students per items matrix where each element represents if the student answered correctly (1) or not (0) to the correspondent item.
`student.avg.ability`	An ability per student matrix where each element represents the ability of a student.
`item.difficulty`	A difficulty per item matrix where each element represents the difficulty of an item.

References

Beheshti, B. and Desmarais, M. C. (2014). Assessing model fit with synthetic vs. real data. Polytechnique Montreal.

See Also

`randGen` for the external students' performance data from which this function can learn.
`minGen` for the external students' performance data from which this function can learn.

avgGen for the external students' performance data from which this function can learn.
dinGen for the external students' performance data from which this function can learn.
irtGen for the external students' performance data from which this function can learn.
poksGen for the external students' performance data from which this function can learn.

Examples

```
#####EXAMPLE 1 : Generation introducing the minimum number of parameters required #####

 #We generate data for 3 students and 5 items.

 result = irtGen(its = 5, sts = 3)

 # We extract the correspondent information

 performance  <- result$results
 performance
       [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    0    0    1
 [2,]    0    0    0    1    1
 [3,]    0    0    0    0    0

 abilities  <- result$student.avg.ability
 abilities
           [,1]       [,2]       [,3]
 [1,] -2.211515 -0.1775592 -0.6316655

 difficulties  <- result$item.difficulty
 difficulties
           [,1]      [,2]      [,3]       [,4]      [,5]
 [1,] 0.3132829 0.1178107 0.5078971 -0.5025616 -0.439553

 #####EXAMPLE 2 : Generation introducing a students' performance matrix as reference #####

 #We generate a students' performance data according to the IRT model.

 reference  <- irtGen(sts = 3, its = 20)$result
 reference
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
 [1,]    1    1    1    1    1    1    1    0    1     1     1     0     0     0     1   ...
 [2,]    0    0    0    1    1    0    1    0    1     0     0     0     0     1     1   ...
 [3,]    0    0    0    1    1    0    1    0    1     0     0     0     1     1     0   ...

 #We generate data for 3 students and 20 items. A reference matrix is provided to learn about
 the students abilities and the items difficulties.

 result  <-  irtGen(its = 20, sts = 3,result = reference)

 # We extract the correspondent information
 performance  <- result$results
 performance
       [,1] [,2] [,3]
 [1,]    1    0    1
 [2,]    1    0    0
 [3,]    0    1    0
```

```
abilities  <- result$student.avg.ability
abilities
      [,1] [,2] [,3]
[1,]    0 -2.4    2

difficulties  <- result$item.difficulty
difficulties
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
[1,]   -1   -1   -1    3    3   -1    3   -3    3    -1    -1    -3    -1     1     1   ...
```

---

minGen *Performance data generation according to a pessimistic approach. Linear model.*

Description

This function generates students' performance data according to a Q-matrix sampling model where more than one skill can be required to succeed an item.
Given a number of skills involved in the set of items, there is a matrix in which each element represents the probability of a student to suceed an item involving a specific skill. It's a students per skills matrix (skills matrix).It is formed by [0-1] values.
So, if an item requires 2 skills, the probability of a student to answer correctly to that item will be the minimum between the probabilities of this student to succeed items involving those skills (skills matrix).

Usage

```
minGen(its, sts, rank, q = NULL, skills = NULL, mean = 0, deviation = 1)
```

Arguments

`its`	Number of items.
`sts`	Number of students.
`rank`	Number of skills the set of items has.
`q`	Q-Matrix representing the skills required to succeed an item. Skills per item matrix. {0,1} values. NULL by default.
`skills`	Matrix representing the probability of each student to answer correctly to an item involving a specific skill. students per skills matrix.[0-1] values. NULL by default.
`mean`	If the skills matrix is not provided to the function, this parameter is the mean used to generate that matrix according to the standard distribution. 0 by default.
`deviation`	If the skills matrix is not provided to the function, this parameter is the standard deviation used to generate that matrix according to the standard distribution. 1 by default.

Value

A list with the following information:

results	Students per items matrix where each element represents if the student answered correctly (1) or not (0) to the correspondent item.
skills.matrix	A student per skill matrix where each element represents the probability of a student to answer correctly to an item that involves the correspondent skill.
q.matrix	The Q-matrix used.

See Also

QgenInc for the external generation of a Q-matrix.
QgenReg for the external generation of a Q-matrix.
extendQ for the external generation of a Q-matrix.
reduceQ for the external generation of a Q-matrix.
skillsGen for the external generation of the skills matrix.

Examples

```
#####EXAMPLE 1 : Generation introducing the minimum number of parameters required #####

 # We generate student performance data about 3 students answering to 5 items and 3 skills.

 result <- minGen(its = 5, sts = 3, rank = 3)

 # We extract the correspondent information

 performance  <- result$results
 performance
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    1    0    1
 [2,]    1    1    0    1    1
 [3,]    1    0    0    0    0

 skills  <- result$skills.matrix
 skills
          [,1]      [,2]      [,3]
 [1,] 0.6760280 0.6686970 0.5695697
 [2,] 0.5326328 0.7144692 0.9995925
 [3,] 0.3629950 0.6258358 0.9310147

 q.matrix  <- result$q.matrix
 q.matrix
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    1    1    1
 [2,]    1    1    0    1    0
 [3,]    1    0    0    1    0

#####EXAMPLE 2 : Generation modeling the mean and the standard deviation used #####

 # We generate student performance data about 3 students answering to 5 items and 3 skills.
 #For the stantard distribution (skills matrix), we use a mean of 0.5 and a standard
 deviation of 0.2.
```

```
result <- minGen(its = 5, sts = 3, rank = 3,mean=0.5,deviation=0.2)

# We extract the correspondent information

performance  <- result$results
performance
      [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    0    0
[2,]    1    0    0    1    0
[3,]    1    1    1    1    0

skills  <- result$skills.matrix
skills
        [,1]      [,2]      [,3]
[1,] 0.7211562 0.6510095 0.7547658
[2,] 0.6853259 0.7352926 0.6736784
[3,] 0.7387059 0.6182576 0.6575972

q.matrix  <- result$q.matrix
q.matrix
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    1    1    1
[3,]    0    0    0    0    0

#####EXAMPLE 3 : Generation introducing a Q-matrix #####

#We generate a Q-Matrix with 3 skills involved.

q.matrix <- QgenInc(num.skills = 3,maxSkillsPerItem=2)
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    1    1    0
[2,]    0    1    0    1    0    1
[3,]    0    0    1    0    1    1

# We generate student performance data about 3 students answering to 6 items, 3 skills and
the Q Matrix we want. For the stantard distribution (skills matrix), we use a mean of 0.5
and a standard deviation of 0.2.

result <- minGen(its = 6, sts = 3, rank = 3,q = q.matrix,mean=0.5,deviation=0.2)

# We extract the correspondent information

performance  <- result$results
performance
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0    1    1    0
[2,]    1    1    1    0    0    1
[3,]    1    1    0    1    1    0
skills  <- result$skills.matrix
skills
        [,1]      [,2]      [,3]
[1,] 0.6868604 0.6387136 0.6409983
[2,] 0.5920339 0.6761624 0.6858127
[3,] 0.7560064 0.6037571 0.5528284
q.matrix  <- result$q.matrix
```

```
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    1    1    0
[2,]    0    1    0    1    0    1
[3,]    0    0    1    0    1    1

#####EXAMPLE 4 : Generation introducing a Q-Matrix and a skills matrix #####

#We generate a Q-Matrix with 3 skills involved.

q.matrix <- QgenInc(num.skills = 3,maxSkillsPerItem=2)
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    1    1    0
[2,]    0    1    0    1    0    1
[3,]    0    0    1    0    1    1

#We generate a skills matrix  skills for 3 students, 3 skills, a mean of 0 and a standard
deviation of 1.

skills  <- skillsGen(sts = 3,mean = 0,deviation = 1,rank = 3)
skills
        [,1]      [,2]      [,3]
[1,] 0.38186877 0.5632612 0.6603963
[2,] 0.02718055 0.2536993 0.2962467
[3,] 0.60535135 0.9708733 0.5421141

# We generate student performance data about 3 students answering to 6 items, 3 skills and
the Q Matrixand skills matrix we want.

result <- minGen(its = 6, sts = 3, rank = 3,q = q.matrix,skills = skills)

performance  <- result$results
performance
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0    1    0    1
[2,]    0    0    0    0    1    1
[3,]    0    1    1    1    0    1

skills  <- result$skills.matrix
skills
        [,1]      [,2]      [,3]
[1,] 0.38186877 0.5632612 0.6603963
[2,] 0.02718055 0.2536993 0.2962467
[3,] 0.60535135 0.9708733 0.5421141

q.matrix  <- result$q.matrix
q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    1    1    0
[2,]    0    1    0    1    0    1
[3,]    0    0    1    0    1    1
```

`oimage` *Grid graphic of a matrix.*

Description

This function represents a matrix under a grid graph. Values closer to 1 will be represented as cells with a color closer to white and values closer to 0 will be cells closer to red.

Usage

```
oimage(data, sort = F, ...)
```

Arguments

data	Matrix to be represented.

Examples

```
x <- randGen(its = 5, sts = 3)$result
 x
       [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    1    0    1
 [2,]    0    1    1    1    0
 [3,]    0    1    0    1    1

 oimage(t(x),xlab='Students',ylab = 'Items')
```

---

poksGen *Generation according to the POKS model.*

Description

In this model, there is a matrix of items per items that represents the dependencies between items. It's called poks matrix and it gives the Partial Orders Knowledge Structure (POKS). Meaning that, if a student answer correctly to an item B, he will have a higher chance to succeed items related to B but easier. That can be modeled as $P(A|B) = p1$. It's a {0,1} matrix where a 1 in the position (i,j) means that suceeding the item i increases the possibilities of succeeding the item j.
When there is no knowledge about an item, the probability of succeeding this item will depends on the student's success rate. This rate can be learned providing a students' performance matrix to the function. Otherwise, it will be generated according to the normal distribution normalized to [0-1] values.

Usage

```
poksGen(its, sts, dependencies = NULL, result = NULL, success.avg = NULL,
  p1, mean = 0, deviation = 1)
```

Arguments

its	Number of items.
sts	Number of students.
dependencies	The poks matrix. NULL by default
success.avg	Matrix with the success rate of each student.
p1	Probability of succeeding an item having succeeded a more difficult one related to it.

`mean` If the results matrix is not provided to the function, the students' success rate matrix will be generated according to the normal function with this parameter as the mean value. 0 by default.

`deviation` If the results matrix is not provided to the function, the students' success rate matrix will be generated according to the normal function with this parameter as the standard deviation value. 1 by default.

`results` A students' performance matrix from which the function learns to generate the students' success rate matrix.

Value

A list with the following information:

`results` Students per items matrix where each element represents if the student answered correctly (1) or not (0) to the correspondent item.

`poks.matrix` The matrix that matches each item with the items that depend on itself.

`success.avg` A matrix with the success average of each student

References

Beheshti, B. and Desmarais, M. C. (2014). Assessing model fit with synthetic vs. real data.

See Also

`randGen` for the external students' performance data from which this function can learn.
`minGen` for the external students' performance data from which this function can learn.
`avgGen` for the external students' performance data from which this function can learn.
`dinGen` for the external students' performance data from which this function can learn.
`irtGen` for the external students' performance data from which this function can learn.
`poksGen` for the external students' performance data from which this function can learn.
`poksMatrixGen` for the external students' performance data from which this function can learn.

Examples

```
#####EXAMPLE 1 : Generation introducing the minimum number of parameters required #####

 #We generate data for 3 students and 5 items. The probability P(A|B) = 0.85

 result <- poksGen(its = 5, sts = 3, p1 = 0.85)

 # We extract the correspondent information

 performance  <- result$results
 performance
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    1    0    0
 [2,]    0    1    1    0    0
 [3,]    1    1    1    1    1

 poks.matrix   <- result$poks.matrix
 poks.matrix
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    0    1    1
```

```
[2,]    0    0    1    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0

success.avg  <- result$success.avg
success.avg
        [,1]     [,2]      [,3]
[1,] 0.2933311 0.419173 0.5056854


#####EXAMPLE 2 : Generation introducing a students' performance matrix as reference and an
external poks matrix#####

#We generate the students' performance data form which we want to learn

reference <- poksGen(its = 20, sts = 3, p1 = 0.85)$result
reference
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
[1,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0  ...
[2,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0  ...
[3,]    0    1    1    0    0    1    1    1    1     1     0     1     1     1     0  ...

#We generate an external poks matrix

poks.matrix  <- poksMatrixGen(its = 8, depNum = 5, trees = 2, indirect.dependencies = TRUE)
poks.matrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    1    0    1    0    0    0    0
[2,]    0    0    1    0    0    0    0    0
[3,]    0    0    0    0    0    0    0    0
[4,]    0    0    0    0    0    0    0    0
[5,]    0    0    0    0    0    1    1    0
[6,]    0    0    0    0    0    0    0    0
[7,]    0    0    0    0    0    0    0    0
[8,]    0    0    0    0    0    0    0    0

#We generate the data according to the parameters generated

result <- poksGen(its = 8, sts = 3 , dependencies = poks.matrix, result = reference,p1 = 0.85)

# We extract the correspondent information

performance  <- result$results
performance
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    0    0    0    0    0    0    0
[2,]    0    0    0    0    0    0    0    0
[3,]    0    0    0    1    1    1    1    1

poks.matrix   <- result$poks.matrix
poks.matrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    1    0    1    0    0    0    0
[2,]    0    0    1    0    0    0    0    0
[3,]    0    0    0    0    0    0    0    0
[4,]    0    0    0    0    0    0    0    0
```

```
[5,]    0    0    0    0    0    1    1    0
[6,]    0    0    0    0    0    0    0    0
[7,]    0    0    0    0    0    0    0    0
[8,]    0    0    0    0    0    0    0    0

success.avg  <- result$success.avg
success.avg
     [,1] [,2] [,3]
[1,]    0    0 0.65
```

---

poksMatrixGen *POKS matrix generation*

Description

This function generates the matrix needed by the POKS model. It gives the Partial Orders Knowledge Structure (POKS). Meaning that, if a student answers correctly to an item B, he'll get a higher chance to succeed items related to B but easier. This can be modeled as $P(A|B) = p1$. It's a {0,1} matrix where a 1 in the position (i,j) means that suceeding the item i increases the possibilities of succeeding the item j.
In order to generate this matrix, parameters as the total number of dependencies, the number of independent trees or the introduction of indirect dependencies can be modeled. A tree is a partial order structure where succeeding an item, the root, increases the probability of answering correctly to the items that are under it with a direct or indirect dependance. The indirect dependance happens when suceeding an item A increases the possibilities of succeeding item B, and item B increases the possibilities of item C. So, indirectly, item A increases the possibilities of item C.

Usage

```
poksMatrixGen(its, depNum, trees, indirect.dependencies = FALSE)
```

Arguments

its	Number of items.
depNum	The total number of dependencies in the poks matrix.
trees	The number of independent trees in the poks matrix.
indirect.dependencies	A boolean determining if the poks matrix contains indirect dependencies. FALSE by default

Value

dependencies	The poks matrix

References

http://cran.r-project.org/web/packages/partitions/index.html

See Also

poksGen for the generation of data according to the POKS model

Examples

```
##### EXAMPLE 1 : POKS matrix without indirect dependencies #####
 #We generate a POKS matrix with 6 items, 2 trees and 4 dependencies.

 poks.matrix <- poksMatrixGen(its = 6, depNum = 4,trees = 2)
 poks.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    0    1    1    0    0    0
 [2,]    0    0    0    0    0    0
 [3,]    0    0    0    0    0    0
 [4,]    0    0    0    0    1    1
 [5,]    0    0    0    0    0    0
 [6,]    0    0    0    0    0    0

##### EXAMPLE 2 : POKS matrix wit indirect dependencies #####
 #We generate a POKS matrix with 8 items, 2 trees and 6 dependencies.

 poks.matrix <- poksMatrixGen(its = 8, depNum = 6,trees = 2,indirect.dependencies = TRUE)
 poks.matrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
 [1,]    0    1    0    1    0    0    0    0
 [2,]    0    0    1    0    0    0    0    0
 [3,]    0    0    0    0    0    0    0    0
 [4,]    0    0    0    0    0    0    0    0
 [5,]    0    0    0    0    0    1    0    1
 [6,]    0    0    0    0    0    0    1    0
 [7,]    0    0    0    0    0    0    0    0
 [8,]    0    0    0    0    0    0    0    0
```

`QgenInc` *Q-matrix generation with a variable number of skills required per item.*

Description

Q-matrix generation where the number of items depends on the maximum number of skills per item chosen. If we define 3 skills involved in the whole set of items, and we want the most difficult item to require 2 skills, we will have a number of items corresponding to all the combinations of 1 skill per item and all the combinations of 2 skills per item. In case we want to set the most difficult item to involve the 3 skills, our Q-matrix would include a last column corresponding to an item involving all the skills.

Usage

```
QgenInc(num.skills, maxSkillsPerItem)
```

Arguments

`num.skills` Number of skills involved.

`maxSkillsPerItem` Maximum number of skills required to succeed an item. It represents the maximum difficulty of an item.

Value

q Q matrix with the information about the skills required to succeed each item. Skills per items matrix.

See Also

minGen for the external Q-Matrix that can be provided to this function .
avgGen for the external Q-Matrix that can be provided to this function.
dinGen for the external Q-Matrix that can be provided to this function.

Examples

```
#We generate a Q-matrix with 4 skills involved and maximum 3 skills involved per item.

 q.matrix <- QgenInc(num.skills = 4, maxSkillsPerItem = 3)
 q.matrix
       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
 [1,]    1    0    0    0    1    1    1    0    0     0     1     1     1     0
 [2,]    0    1    0    0    1    0    0    1    1     0     1     1     0     1
 [3,]    0    0    1    0    0    1    0    1    0     1     1     0     1     1
 [4,]    0    0    0    1    0    0    1    0    1     1     0     1     1     1
```

QgenReg *Q-matrix generation with the same number of skills required per item.*

Description

Q-matrix with dimensions corresponding to all the combinations of items involving the number of skills per item defined. In this function, all the items have the the same number of skills involved.

Usage

```
QgenReg(num.skills, skillsPerItem)
```

Arguments

num.skills Number of skills involved.

skillsPerItem Number of skills per item.

Value

q Q matrix with the information about the skills required to succeed each item. Skills per items matrix.

See Also

minGen for the external Q-Matrix that can be provided to this function.
avgGen for the external Q-Matrix that can be provided to this function.
dinGen for the external Q-Matrix that can be provided to this function.

Examples

```
#We generate a Q-matrix with 4 skills involved and 2 skills involved per item.

 q.matrix <- QgenReg(num.skills = 4, skillsPerItem = 2)
 q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    1    1    1    0    0    0
 [2,]    1    0    0    1    1    0
 [3,]    0    1    0    1    0    1
 [4,]    0    0    1    0    1    1
```

---

randGen *Performance data generation according to independent probabilities of success. Linear model.*

Description

This function generates students' performance data without considering neither a skills per student matrix nor a Q-Matrix.
All the students are answering to all the items according to a [0-1] students per items matrix where each element represents the probability of the student i to succeed the item j. The mentioned matrix (students x items) is generated by default according to the normal distribution with a mean of 0 and a standard deviation of 1.
The user can modify those parameters. However, the user can also introduce the matrix by himself.

Usage

```
randGen(its, sts, success.prob.matrix = NULL, mean = 0, deviation = 1)
```

Arguments

`its`	Number of items.
`sts`	Number of students.
`success.prob.matrix`	Matrix with dimension students*items. Each value represents the probability of a student i to success the item j.[0-1] matrix. NULL by default
`mean`	If the success.prob.matrix is not provided to the function, this parameter is the mean used to generate that matrix according to the standard distribution (normalized afterwards between 0 and 1). 0 by default.
`deviation`	If the success.prob.matrix is not provided to the function, this parameter is the normal deviation used to generate that matrix according to the standard distribution (normalized afterwards between 0 and 1). 1 by default.

Value

A list with the following information:

`results`	Students per items matrix where each element represents if the student answered correctly (1) or not (0) to the correspondent item.
`success.prob.matrix`	A student per item matrix where each element represents the probability of a student to answer correctly to the correspondent item.

See Also

successProbGen for the external generation of a student per item probability of success matrix.

Examples

```
##### EXAMPLE 1 : Generation just introducing the number of items and students #####

 # We generate student performance data about 3 students answering to 5 items

 result <- randGen(its = 5, sts = 3)

 # We extract the correspondent information

 performance <- result$results
 performance
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    1    1    1    0
 [2,]    1    1    1    0    1
 [3,]    1    0    1    0    0

 success.probabilities <- result$success.prob.matrix
 success.probabilities
        [,1]       [,2]      [,3]      [,4]      [,5]
 [1,] 0.01231753 0.90747166 0.4095728 0.9999342 0.2118582
 [2,] 0.68874955 0.81822425 0.6833268 0.8773570 0.7834774
 [3,] 0.73083892 0.02549496 0.7293477 0.2291466 0.4035640


 ###EXAMPLE 2 : Generation modeling the mean and standard deviation of the standard
 distribution##

 # We generate student performance data about 3 students answering to 5 items with a
 mean of 0.5 and a standard deviation of 0.2.

 result <- randGen(its = 5, sts = 3,mean = 0.5,deviation=0.2)

 # We extract the correspondent information

 performance  <- result$results
 performance
      [,1] [,2] [,3] [,4] [,5]
 [1,]    1    1    1    1    1
 [2,]    1    1    1    1    0
 [3,]    1    1    0    1    0

 success.probabilities <- result$success.prob.matrix
 success.probabilities
        [,1]      [,2]      [,3]      [,4]      [,5]
 [1,] 0.8237951 0.6739213 0.7073844 0.7008126 0.7444546
 [2,] 0.5642406 0.6735758 0.7183799 0.6911244 0.5717511
 [3,] 0.7246194 0.8351532 0.6331505 0.6782793 0.6518936


###EXAMPLE 3 : Generation providing the student per item success probability matrix##
```

```
# We first generate the desired matrix

prob.success  <- successProbGen(its = 5, sts = 3)
prob.success
        [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.3788828 0.6811485 0.7916251 0.4924259 0.7644050
[2,] 0.6076803 0.7911027 0.9229351 0.9763337 0.4992891
[3,] 0.9118014 0.9351249 0.7218492 0.7246330 0.8756847

# We generate student performance data about 3 students answering to 5 items according
to a given matrix.

result <- randGen(its = 5, sts = 3,success.prob.matrix = prob.success)

# We extract the correspondent information

performance <- result$results
performance
      [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    0    1
[2,]    0    1    1    1    0
[3,]    1    1    1    0    1

success.probabilities  <- result$success.prob.matrix

# We expect the same as provided

success.probabilities
        [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.3788828 0.6811485 0.7916251 0.4924259 0.7644050
[2,] 0.6076803 0.7911027 0.9229351 0.9763337 0.4992891
[3,] 0.9118014 0.9351249 0.7218492 0.7246330 0.8756847
```

`reduceQ` *Q-matrix reduction.*

Description

This function reduces the size of a Q-Matrix randomly extracting the columns (number of items) needed.

Usage

```
reduceQ(q, numberOfItems)
```

Arguments

`q`	Q-matrix to reduce in size.
`numberOfOtems`	Number of items(columns) of the final Q-Matrix

Value

`q`	Q matrix with the information about the skills required to succeed each item. Skills per items matrix.

See Also

QgenReg for the generation of a Q-Matrix .
QgenInc for the generation of a Q-Matrix .
extendQ to modify the size of the a Q-matrix.

Examples

```
#We generate a Q-matrix with 4 skills involved and 3 skills as maximum involved per item.

 q.matrix <- QgenInc(num.skills = 4, maxSkillsPerItem = 3)
 q.matrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
 [1,]    1    0    0    0    1    1    1    0    0     0     1     1     1     0
 [2,]    0    1    0    0    1    0    0    1    1     0     1     1     0     1
 [3,]    0    0    1    0    0    1    0    1    0     1     1     0     1     1
 [4,]    0    0    0    1    0    0    1    0    1     1     0     1     1     1

 # We generate a matrix of just 10 items

 new.qmatrix <- reduceQ(q = q.matrix, numberOfItems = 10)
 new.qmatrix
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    0    0    1    1    1    0    0    1     1
 [2,]    0    1    1    0    1    0    1    0    0     0
 [3,]    1    0    0    1    1    0    1    1    1     1
 [4,]    0    1    1    1    0    1    0    1    1     1
```

---

skillsGen *Skills per student matrix generation according to the standard distribution.*

Description

This function generates a skills per students matrix where each value represents the probability of a student i to answer correctly to an item involving the skill j.

Usage

```
skillsGen(sts, mean, deviation, rank)
```

Arguments

sts	Number of students.
mean	Mean of the standard distribution.
deviation	Standard deviation of the standard distribution. It represents a noise while assigning skills to each student.
rank	Number of skills the set of items has.

Value

skills	The skills per student matrix

See Also

minGen for the generation of a skills matrix that can be provided to that function.
avgGen for the generation of a skills matrix that can be provided to that function.

Examples

```
#We generate a skills matrix for 3 students, involving 4 skills, with a mean of 0 and
a standard deviation of 1.

 skills  <- skillsGen(sts = 3, mean = 0, deviation = 1, rank = 4)
 skills
         [,1]      [,2]       [,3]      [,4]
 [1,] 0.2379563 0.4103992 0.97868219 0.1400159
 [2,] 0.7146010 0.7078044 0.59517022 0.4639706
 [3,] 0.9454237 0.9641969 0.03825092 0.8487321
```

---

successProbGen — *Students per items matrix generation according to the standard distribution.*

Description

This function generates a matrix with dimensions students*items where each value represents the probability of a student i to success an item j.

Usage

```
successProbGen(its, sts, mean, deviation)
```

Arguments

its	Number of items.
sts	Number of students.
mean	Mean of the standard distribution.
deviation	Standard deviation of the standard distribution. It represents a noise while assigning skills to each student.
rank	Number of skills the set of items has.

Value

succes.prob	The skills matrix

See Also

randGen for the generation of a success probability matrix that can be provided to that function.

Examples

```
#We generate a success probabilty matrix for 3 students and 5 items, with a mean of 0
and a standard deviation of 1.

 success.prob <- successProbGen(its = 5, sts = 3, mean = 0, deviation = 1)
 success.prob
         [,1]      [,2]      [,3]      [,4]      [,5]
 [1,] 0.3657776 0.1630787 0.5882404 0.9314930 0.1885536
 [2,] 0.8937646 0.6969860 0.9854258 0.5449783 0.7714228
 [3,] 0.9655305 0.5075924 0.1038589 0.4710194 0.7395491
```

Index

*Topic **package**
SDG-package, 1

avgGen, 2, *15*, *21*, *25*, *30*

barComparison, 6

compare, 7

dinGen, 8, *15*, *21*, *25*

extendQ, *3*, *9*, 12, *17*, *29*

irtGen, 13, *15*, *21*

minGen, *14*, 16, *21*, *25*, *30*

oimage, 19

poksGen, *15*, 20, *21*, *23*
poksMatrixGen, *21*, 23

QgenInc, *3*, *9*, *12*, *17*, 24, *29*
QgenReg, *3*, *9*, *12*, *17*, 25, *29*

randGen, *14*, *21*, 26, *30*
reduceQ, *3*, *9*, *12*, *17*, 28

SDG (SDG-package), 1
SDG-package, 1
skillsGen, *3*, *9*, *17*, 29
successProbGen, *27*, 30