Educational Data Synthesizer

Hoang-Trieu Trinh, Behzad Beheshti, Michel C. Desmarais

2016-04-30

Introduction

In general, generating data from a given set of parameters under a specified model's assumptions is straightforward, as long as the following two conditions are satisfied:

- *condition 1* : All the necessary parameters are given.
- *condition 2* : Information inferred from the given parameters' values does not conflict with itself (user input is consistent).

However, checking for these conditions is not always a straightforward task. Namely, difficulties may arise from the Educational Data Mining literature, where parameters are usually organised in a complex manner. Some of them are shared between different models (e.g. average success rate, student variance, number of concepts, etc), the collection of all parameters can also be arranged into a hierarchical structure in the sense that some parameters can be used to generate some others, as oppose to a set of parameters that directly generates data.

For example, with Non-negative Matrix Factorization models, Q-matrix and Skill matrix are parameters that directly generate data, while these two parameters can be generated using parameters at lower level such as number of students, number of items, or number of concepts. A sufficient set of parameters to generate data in this case can be {Q-matrix, Skill matrix} or {Q-matrix, number of students}. For the former set to be consistent, number of columns of Q-matrix must equate the number of rows of Skill matrix because both represent number of concepts.

One can conclude from the above observation that there are many different ways to generate data depending on which model is being acquired and which parameters are given (assuming that they all satisfy *condition 1* and *condition 2*). At this point it is important to stress that for different sufficient and consistent sets of parameters, even with respect to a single model M, we must expect different amounts of variance in the generated data. The reason is, with a set of parameters at lower level, it requires more intermediate generating steps before reaching the final one (generating data), thus produces more variability in the corresponding generated data.

R package `edmsyn` provides users a simple framework that conveniently handles all the situations above while generating data, including checking for *condition 1* and *condition 2*. It also automates the process of learning parameters from raw data, modifying and using this information to create synthetic data from 10 different models. This document is intended to give a quick and thorough tutorial on using `edmsyn`.

Loading the package

```
library(edmsyn)
```

Vector `edmconst$ALL.MODELS` loaded at the start of this package contains the names of all available models.

```
ALL.MODELS <- edmconst$ALL.MODELS
ALL.MODELS
```

```
##  [1] "exp"     "irt"     "poks"    "dina"    "dino"    "lin.avg" "nmf.con"
##  [8] "nmf.dis" "nmf.com" "bkt"
```

Class context and function pars()

Since some of the parameters are shared between different models, edmsyn did not regard the collection of all parameters from 10 models separately, but jointly as different aspects describing a single instance of reality. For example, the vector of discrimination factors of all items is a parameter belongs to the IRT-2pl model, which is not the case for the parameter Q-matrix, however, edmsyn introduces the class context where these two parameters can co-exist in a single object and can be utilized according to user purpose.

Function pars() produce an object of class context.

```
p <- pars(students = 15, items = 20)
class(p)
```

```
## [1] "context"
```

By the assignment `p <- pars(students = 20, items = 20)` there is currently no other information in the context p except students, items, and default.vals.

```
print(p)
```

```
## Activated information in the context:
## [1] "items"        "students"     "default.vals"
```

p - a context object, is essentially a list object with different components. default.vals is the component that is always available (activated) in any context object, it is an environment containing the default values for some of the parameters. Whenever one of these parameters is needed without user idicated input, the corresponding value will be fetched from default.vals and then loaded into the context.

```
class(p$default.vals)
```

```
## [1] "defaults"
```

```
names(p$default.vals)
```

```
##  [1] "min.ntree"        "bkt.mod"          "L.st.var"
##  [4] "abi.mean"         "student.var"      "min.depth"
##  [7] "alpha.p"          "trans"            "time"
## [10] "p.min"            "density"          "avg.success"
## [13] "bkt.slip.st.var"  "bkt.guess.st.var" "abi.sd"
## [16] "min.it.per.tree"  "S.st.var"         "per.item"
## [19] "alpha.c"
```

```
p$default.vals$avg.success
```

```
## [1] 0.5
```

```
p$avg.success
```

```
## NULL
```

At some point in the future, if data need to be generated from `p` by a model that requires `avg.success`, in case this parameter had not been inputted by the user, the value of `0.5` would be used.

We can change these default values by using the `init` function

```
p_ <- pars(default.vals = default(avg.success = 0.6))
p_$default.vals$avg.success
```

```
## [1] 0.6
```

Function `pars` can also be used to update an available `context`, for example, we can change the number of students in `p` from 15 to 20, add the *number of concepts* information into `p`, and delete the *number of items* information.

```
p <- pars(p, concepts = 5, students = 20, items = NULL)
p
```

```
## Activated information in the context:
## [1] "students"     "default.vals" "concepts"
```

```
# Recover items = 20 for later use
p <- pars(p, items = 20)
p
```

```
## Activated information in the context:
## [1] "students"     "default.vals" "concepts"     "items"
```

`pars` also automatically figures out obtainable parameters at lower level whenever a parameter is activated. For example, dimensions of `M` imply the number of concepts and the number of students. Thus, if a context is supplied with `M`, parameters `concepts` and `students` (and some other obtainable parameters) are activated.

```
p_ <- pars(M = matrix(1, 7, 30))
p_
```

```
## Activated information in the context:
## [1] "M"            "default.vals" "concepts"     "students"
## [5] "concept.exp"
```

```
p_$concepts
```

```
## [1] 7
```

While assembling input parameters from users, `pars` will check for their consistency (*condition 2*) [1]. Since `p` is a context where there are 20 question items, a Q-matrix with 30 rows is incompatible.

```
p <- pars(p, Q = matrix(1,30,5))
```

```
## Error in down.stream(new.pars): 'items' receives different values at once
```

[1]Currently `edmsyn` is able to detect whether a parameter is receiving different values, or if it violates some bounds indicated by users

In short, the introduction of class `context` and function `pars` is mostly for convenience in the following cases:

- A user is working across many different models with some shared parameters.
- A user wants to study data generation from different sufficient sets of parameters. Without having to use many functions to handle each case separately, he/she will only need a function to deal with objects of a single class `context`.

Get the value of a parameter from a context by `get.par`

We can access to components of a `context` object just like accessing to components of a list. `edmsyn` provides an alternative approach to this by the function `get.par`

```
p$students
```

```
## [1] 20
```

```
get.par("students", p)
```

```
## $value
## [1] 20
##
## $context
## Activated information in the context:
## [1] "students"     "default.vals" "concepts"     "items"
```

The advantage of `get.par` is that, it gives result even when the parameter is not available, as long as the parameter is possible to be generated. For example, if we want to produce the value of the skill mastery matrix (`M`) from `p`, the conventional operator `$` will return `NULL`, while `get.par` finds a way to generate `M` from `concepts` and `students`. Set the argument `progress = TRUE` to see how `get.par` does it.

```
p$M
```

```
## NULL
```

```
M_ <- get.par("M", p, progress = TRUE)
```

```
## Generate M from c("students", "concept.exp")
## Generate concept.exp from concepts
```

```
M_
```

```
## $value
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    0    1    0    0    0    0    0    0     0     0     1     0
## [2,]    0    0    0    1    0    0    0    0    0     0     0     0     0
## [3,]    1    1    0    0    1    0    0    1    1     1     1     1     1
## [4,]    0    1    1    0    1    0    1    1    1     0     1     0     1
## [5,]    1    0    0    0    0    0    0    0    0     0     0     1     0
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]     0     0     0     1     0     0     0
## [2,]     0     0     0     0     0     0     0
## [3,]     0     1     0     0     1     0     1
## [4,]     0     1     0     1     1     1     1
## [5,]     0     0     0     0     0     0     0
##
## $context
## Activated information in the context:
## [1] "students"     "default.vals" "concepts"     "items"
## [5] "concept.exp"  "M"
```

As can be seen, the progress involves generating an intermediate parameter called `concept.exp` before generating `M`. `concept.exp` is a vector of the expected mastery rates for 5 concepts in `p`. By using `get.par`, we can examine the values of these intermediate generations.

```
M_$context$concept.exp
```

```
## [1] 0.15 0.05 0.60 0.65 0.10
```

For example, looking at the above `concept.exp`, it is expected that there is about 15 percent of the students who mastered the first concept in `p`. In fact, due to the abundant availability of parameters co-existing in a context, generating `concept.exp` is just one amongst many other ways by which `get.par` can reach `M`, the reason why it turns out to be this way will be discussed in later parts.

If we run `get.par` to obtain `M` again, in general the result will be different. This is because of the fact that each time doing so, another probabilistic generation will be carried out, thus we receive a different `concept.exp`, and after that a different `M`.

```
identical(M_$value, get.par("M", p)$value)
```

```
## [1] FALSE
```

`get.par` can detect whether there is enough information to generate the required parameters. For example, Q-matrix needs the number of concepts to be defined, without the number of concepts, there is no way Q-matrix can be derived.

```
p_ <- pars(students = 20, items = 15)
get.par("Q", p_)
```

```
## Error in up.stream(target, pars, FALSE, progress): Cannot reach 'Q' since 'concepts' is missing
```

One way to fix this is supplying `M` to the context, which essentially contains the *number of concepts* information. By this, the process becomes possible

```
p_ <- pars(p_, M = matrix(1, 4, 20))
get.par("Q", p_)
```

```
## $value
##       [,1] [,2] [,3] [,4]
##  [1,]    1    1    1    0
##  [2,]    1    1    0    0
##  [3,]    0    1    0    0
##  [4,]    0    1    1    0
##  [5,]    1    1    1    0
##  [6,]    1    1    0    1
##  [7,]    1    0    0    1
##  [8,]    0    0    1    1
##  [9,]    1    1    0    1
## [10,]    0    1    1    0
## [11,]    1    0    1    0
## [12,]    1    0    1    1
## [13,]    1    0    0    0
## [14,]    0    1    0    1
## [15,]    0    0    1    0
##
## $context
## Activated information in the context:
## [1] "items"        "students"     "default.vals" "M"
## [5] "concepts"     "concept.exp"  "Q"
```

Generate data from a context using `gen`

Parameters activated in a single context are ensured non-conflict while being put together by `pars` (*condition 2*). If a context satisfies *condition 1*, there will be a way to generate data from it. `edmsyn` provides function `gen` that can check for *condition 2* and generates data with the specified model. For example, the following code generates data from `p` by POKS model (Partial Order Knowledge Structure model) twice:

```
poks.data <- gen("poks", p, n = 2, progress = TRUE)
```

```
## Generate poks from c("students", "po", "avg.success", "alpha.c", "alpha.p", "p.min")
## Generate po from c("items", "min.ntree", "max.ntree", "min.depth", "max.depth", "density", "min.it.p
## Generate min.ntree from default.vals
## Generate max.ntree from items
## Generate min.depth from default.vals
## Generate max.depth from items
## Generate density from default.vals
## Generate min.it.per.tree from default.vals
## Generate max.it.per.tree from items
## Generate trans from default.vals
## Generate avg.success from default.vals
## Generate alpha.c from default.vals
## Generate alpha.p from default.vals
## Generate p.min from default.vals
```

```
poks.data
```

```
## [[1]]
## Activated information in the context:
##  [1] "students"        "default.vals"    "concepts"
##  [4] "items"           "min.ntree"       "min.depth"
##  [7] "density"         "min.it.per.tree" "trans"
## [10] "avg.success"     "alpha.c"         "alpha.p"
## [13] "p.min"           "max.ntree"       "max.depth"
## [16] "max.it.per.tree" "po"              "poks"
##
## [[2]]
## Activated information in the context:
##  [1] "students"        "default.vals"    "concepts"
##  [4] "items"           "min.ntree"       "min.depth"
##  [7] "density"         "min.it.per.tree" "trans"
## [10] "avg.success"     "alpha.c"         "alpha.p"
## [13] "p.min"           "max.ntree"       "max.depth"
## [16] "max.it.per.tree" "po"              "poks"
```

As can be seen, gen returns a list of two components, correspond to two generations required by option `n = 2`. Each of the two is a context. `edmsyn` views the generated data as *a part of the context*. In fact, we can consider *data* as a parameter at highest level. In each of the generated context, there is a component whose name is the same as the specified model `poks`, this component contains the generated data.

```
poks.data[[1]]$poks
```

```
## $R
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    1    0    0    0    1    0    1    0    0     0     1     1     0
##  [2,]    1    0    0    1    0    0    0    1    1     0     1     0     1
##  [3,]    1    1    1    1    1    0    1    0    0     1     0     1     0
##  [4,]    1    1    1    0    1    0    1    1    1     1     1     1     1
##  [5,]    0    1    1    0    1    1    1    1    0     0     1     1     0
##  [6,]    0    0    0    1    1    0    0    0    0     0     0     1     0
##  [7,]    1    0    0    1    1    1    0    1    1     0     0     0     0
##  [8,]    1    1    1    1    1    1    1    1    0     1     1     0     0
##  [9,]    0    0    0    1    1    0    0    0    0     0     0     1     0
## [10,]    1    0    1    0    1    1    1    0    1     1     1     1     1
## [11,]    1    1    1    1    1    1    1    1    0     1     1     1     1
## [12,]    0    0    0    1    1    0    0    0    0     0     0     1     0
## [13,]    1    1    1    1    1    0    1    0    0     1     1     1     1
## [14,]    0    0    1    0    1    0    0    0    1     1     1     0     1
## [15,]    0    1    1    1    0    1    1    1    0     1     1     0     1
## [16,]    0    0    0    1    1    0    0    0    0     0     0     1     0
## [17,]    0    1    0    0    0    1    0    1    0     1     1     0     1
## [18,]    0    0    1    1    1    0    1    0    0     1     0     1     1
## [19,]    1    1    1    1    1    1    1    1    1     1     1     1     1
## [20,]    0    0    0    1    1    0    1    0    0     0     0     1     1
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20]
##  [1,]     1     0     0     1     0     0     1
##  [2,]     1     1     1     1     1     1     0
##  [3,]     0     0     0     1     0     1     1
##  [4,]     1     1     1     1     1     1     0
##  [5,]     1     1     1     0     0     1     1
##  [6,]     1     0     0     1     1     0     0
##  [7,]     1     1     0     1     1     0     1
##  [8,]     0     1     1     1     1     1     0
##  [9,]     1     0     0     1     1     0     0
## [10,]     1     1     1     1     1     0     1
## [11,]     1     1     1     1     1     1     1
## [12,]     1     1     0     1     1     0     0
## [13,]      1      1      1      1      1      0      1
## [14,]      1      0      1      1      1      0      1
## [15,]      0      1      1      1      1      1      0
## [16,]      0      1      0      1      0      0      0
## [17,]      0      1      0      0      1      1      0
## [18,]      1      0      1      1      1      0      1
## [19,]      1      1      1      1      1      1      1
## [20,]      1      0      1      1      1      0      0
##
## $alpha.c
## [1] 0.25
##
## $alpha.p
## [1] 0.25
##
## $p.min
## [1] 0.5
```

Looking at the result, the `poks` component is a list with component `R` being the response matrix, and other components representing other information that is needed for the learning process. This point will be explained in later sections.

Since data is considered to be just another parameter in a context, we can actually use `get.par` to generate data. In fact, `gen` is simply a wrapper of `get.par` with an additional feature `n`, where users can specify how many times the process should be repeated.

```
get.par("poks", p)
```

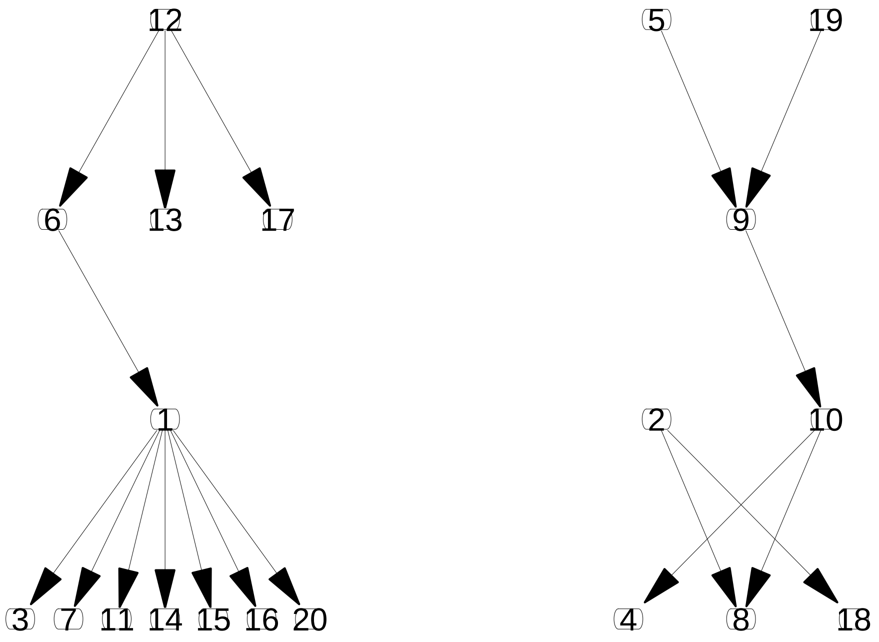
```
## $value
## $value$R
##        [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    1    0    0    1    0    1    1    0    0     1     0     0     1
##  [2,]    0    0    1    0    0    1    0    0    1     0     1     1     0
##  [3,]    1    1    0    0    1    0    0    1    0     1     1     0     0
##  [4,]    0    0    1    1    1    1    0    1    1     0     0     1     1
##  [5,]    0    1    1    0    1    1    0    0    0     1     0     0     0
##  [6,]    1    1    0    1    1    0    0    1    0     1     0     1     1
##  [7,]    1    1    0    0    1    0    1    1    1     1     1     1     1
##  [8,]    1    1    1    0    1    0    1    1    1     1     1     1     1
##  [9,]    1    1    1    0    1    1    0    1    0     1     1     0     0
## [10,]    1    1    0    0    1    0    1    1    1     1     1     1     0
## [11,]    1    0    1    1    0    0    1    0    0     1     0     0     0
## [12,]    1    1    1    1    1    1    1    1    1     1     1     1     1
## [13,]    1    1    0    1    1    0    1    1    1     1     0     1     1
## [14,]    0    1    1    1    0    0    1    0    1     1     1     1     1
## [15,]    0    1    0    0    1    0    0    1    1     1     1     0     0
## [16,]    0    0    0    0    1    0    0    0    1     0     0     0     0
## [17,]    1    1    1    1    0    1    0    0    1     0     1     1     0
## [18,]    1    1    0    0    1    0    0    1    0     1     1     1     0
## [19,]    1    1    1    1    0    0    1    1    1     1     0     0     1
## [20,]    1    1    1    1    1    1    0    0    1     1     1     1     0
##        [,14] [,15] [,16] [,17] [,18] [,19] [,20]
##  [1,]      1     0     1     0     0     1     1
##  [2,]      0     1     0     0     0     0     0
##  [3,]    1    1    0    1    1    0    0
##  [4,]    1    0    0    1    0    0    1
##  [5,]    0    0    0    0    0    0    0
##  [6,]    0    0    0    0    0    1    1
##  [7,]    1    1    1    1    1    0    1
##  [8,]    1    1    1    1    1    1    1
##  [9,]    1    1    0    0    1    0    0
## [10,]    1    1    1    1    1    0    1
## [11,]    1    0    1    1    0    0    1
## [12,]    1    1    1    1    1    1    1
## [13,]    1    0    0    1    1    0    1
## [14,]    0    1    1    1    0    1    0
## [15,]    1    1    0    1    1    0    0
## [16,]    0    0    0    1    0    0    0
## [17,]    1    1    0    1    1    0    0
## [18,]    1    1    1    1    1    0    0
## [19,]    1    1    1    1    0    1    1
## [20,]    1    1    1    1    1    1    1
##
## $value$alpha.c
## [1] 0.25
##
## $value$alpha.p
## [1] 0.25
##
## $value$p.min
## [1] 0.5
##
##
## $context
## Activated information in the context:
##  [1] "students"         "default.vals"     "concepts"
##  [4] "items"            "min.ntree"        "min.depth"
##  [7] "density"          "min.it.per.tree"  "trans"
## [10] "avg.success"      "alpha.c"          "alpha.p"
## [13] "p.min"            "max.ntree"        "max.depth"
## [16] "max.it.per.tree"  "po"               "poks"
```

Following is two more examples on using `gen`. In the first one, `gen` will generate data by POKS model again, but from a different context where user have more control on the Partial Order structure of items. In the second one, `gen` generates data by DINA model (Deterministic Input Noisy And model).

Example 1

Suppose this time we want the Partial Order structure of items to have two connected components, each with height 3 and no transitive link between items, this can be done by updating `p` with some more parameters. To visualize this structure, whose dependency matrix will be component `po` in the context returned by `gen`, we can use function `viz`.

```r
p <- pars(p, min.depth = 3, max.depth = 3, min.ntree = 2, max.ntree = 2, trans = FALSE)
poks.data <- gen("poks", p)
v <- viz(poks.data$po)
```



v contains analysed information about the structure. Its first component is identical to po, the second one is a list with equal number of components to the number of connected components of the structure po represents. Each of these components is in turn a list with first component being the corresponding dependency matrix, and sencond component represents how many items are there on each level of depth.

```
print(v)
```

```
## $ks
##    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## 1  0 0 1 0 0 0 1 0 0  0  1  0  0  1  1  1  0  0  0  1
## 2  0 0 0 0 0 0 0 1 0  0  0  0  0  0  0  0  0  1  0  0
## 3  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 4  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 5  0 0 0 0 0 0 0 0 1  0  0  0  0  0  0  0  0  0  0  0
## 6  1 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 7  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 8  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 9  0 0 0 0 0 0 0 0 0  1  0  0  0  0  0  0  0  0  0  0
## 10 0 0 0 1 0 0 0 1 0  0  0  0  0  0  0  0  0  0  0  0
## 11 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 12 0 0 0 0 0 1 0 0 0  0  0  0  1  0  0  0  1  0  0  0
## 13 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 14 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 15 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 16 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 17 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 18 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 19 0 0 0 0 0 0 0 0 1  0  0  0  0  0  0  0  0  0  0  0
## 20 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
##
## $comp
## $comp[[1]]
## $comp[[1]]$matrix
##    12 6 13 17 1 3 7 11 14 15 16 20
## 12  0 1  1  1 0 0 0  0  0  0  0  0
## 6   0 0  0  0 1 0 0  0  0  0  0  0
## 13  0 0  0  0 0 0 0  0  0  0  0  0
## 17  0 0  0  0 0 0 0  0  0  0  0  0
## 1   0 0  0  0 0 1 1  1  1  1  1  1
## 3   0 0  0  0 0 0 0  0  0  0  0  0
## 7   0 0  0  0 0 0 0  0  0  0  0  0
## 11  0 0  0  0 0 0 0  0  0  0  0  0
## 14  0 0  0  0 0 0 0  0  0  0  0  0
## 15  0 0  0  0 0 0 0  0  0  0  0  0
## 16  0 0  0  0 0 0 0  0  0  0  0  0
## 20  0 0  0  0 0 0 0  0  0  0  0  0
##
## $comp[[1]]$level.sizes
## [1] 1 3 1 7
##
##
## $comp[[2]]
## $comp[[2]]$matrix
##    5 19 9 2 10 4 8 18
## 5  0  0 1 0  0 0 0  0
## 19 0  0 1 0  0 0 0  0
## 9  0  0 0 0  1 0 0  0
## 2  0  0 0 0  0 0 1  1
## 10 0  0 0 0  0 1 1  0
## 4  0  0 0 0  0 0 0  0
## 8  0  0 0 0  0 0 0  0
## 18 0  0 0 0  0 0 0  0
##
## $comp[[2]]$level.sizes
## [1] 2 1 2 3
```

Example 2

```
dina.data <- gen("dina", p, progress = TRUE)
```

```
## Generate dina from c("Q", "M", "slip", "guess")
## Generate Q from c("items", "concepts")
## Generate M from c("students", "concept.exp")
## Generate concept.exp from concepts
## Generate slip from items
## Generate guess from items
```

```
dina.data
```

```
## Activated information in the context:
##  [1] "students"     "default.vals" "concepts"     "items"
##  [5] "min.ntree"    "max.ntree"    "trans"        "min.depth"
##  [9] "max.depth"    "Q"            "concept.exp"  "M"
## [13] "slip"         "guess"        "dina"
```

As reported by **gen**, unlike the previous case, `M` is now generated from three parameters **students**, **skill.space**, and **skill.dist** instead of **concept.exp** and **students**. The reason for this particular situation is that, `skill.space` and `skill.dist` are formal parameters of DINA model, so they should be used in the process of generating data. Again, we can access this data by referring to component `dina` of the generated context. Besides the response matrix, `dina.data$dina` also have another component `Q`, this is because of the fact that under DINA's view, a response matrix without its corresponding Q-matrix is incomplete.

```
dina.data$dina
```

```
## $R
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    0    1    1    1    1    1    1    1    1     1     1     1     1
##  [2,]    0    0    1    0    1    0    0    1    0     0     0     0     1
##  [3,]    0    1    1    1    0    1    1    0    1     0     1     1     1
##  [4,]    1    1    1    1    1    0    1    1    1     1     1     1     1
##  [5,]    1    0    1    1    0    1    0    0    1     1     1     0     0
##  [6,]    0    0    0    1    1    1    0    1    1     1     1     1     1
##  [7,]    1    0    1    1    0    0    0    0    0     0     0     0     1
##  [8,]    0    0    1    1    0    0    0    0    0     0     1     0     0
##  [9,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [10,]    0    0    0    0    1    0    0    0    0     0     1     0     0
## [11,]    1    1    1    1    1    0    1    1    1     1     1     1     1
## [12,]    0    1    0    0    0    0    0    0    0     0     0     0     0
## [13,]    1    1    1    1    0    1    0    0    1     0     1     0     1
## [14,]    0    1    0    0    1    0    1    1    1     1     0     1     0
## [15,]    1    1    1    1    1    1    1    1    1     1     1     1     1
## [16,]    0    0    0    1    0    1    0    1    1     0     0     0     0
## [17,]    1    1    1    1    1    0    1    1    1     1     1     0     1
## [18,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [19,]    0    1    1    1    0    1    1    0    1     0     0     0     1
## [20,]    0    0    0    0    0    0    0    0    0     1     0     0     0
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20]
##  [1,]     1     1     1     1     1     1     1
##  [2,]     1     0     0     0     0     0     0
##  [3,]     1     1     1     0     0     1     1
##  [4,]     1     1     1     0     0     1     0
##  [5,]     0     1     1     1     1     1     1
##  [6,]     1     0     0     1     1     0     0
##  [7,]     0     1     0     0     0     0     0
##  [8,]     0     0     0     0     1     0     0
##  [9,]     0     0     0     0     0     0     0
## [10,]     0     0     0     0     0     0     0
## [11,]     1     1     1     1     1     1     1
## [12,]     0     0     0     0     0     0     0
## [13,]     1     1     1     1     1     1     0
## [14,]     1     0     0     0     0     1     0
## [15,]     1     1     1     1     0     1     1
## [16,]     0     0     0     0     0     0     0
## [17,]     0     1     1     1     1     1     1
## [18,]     0     0     0     0     0     0     0
## [19,]     1     0     1     1     1     1     1
## [20,]     0     0     0     1     0     0     1
##
## $Q
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1    0    1    1    0
##  [2,]    0    1    0    1    0
##  [3,]    0    0    1    1    0
##  [4,]    0    1    1    1    1
##  [5,]    0    0    0    1    0
##  [6,]    0    1    0    0    1
##  [7,]    0    1    1    1    0
##  [8,]    1    0    1    0    1
##  [9,]    1    0    1    0    0
## [10,]    1    0    0    0    1
## [11,]    1    1    1    1    0
## [12,]    0    1    1    1    0
## [13,]    1    1    1    0    0
## [14,]    1    0    1    0    1
## [15,]    1    1    1    0    0
## [16,]    0    0    0    1    1
## [17,]    1    1    1    1    0
## [18,]    1    1    0    1    0
## [19,]    1    1    1    1    1
## [20,]    1    1    0    1    1
```

`gen` only allow one model and one context at a time, we can save time generating data across different models and contexts using `gen.apply`. Setting the argument `multiply` to `TRUE` or `FALSE` decides what kind of matching will be made between models and contexts.

```
dat.1 <- gen.apply(ALL.MODELS, list(p1 = p,p2 = p_), multiply = FALSE, n = 5)
dat.1
```

```
##            01      02      03      04      05
## exp.p1     List,14 List,14 List,14 List,14 List,14
## irt.p2     List,10 List,10 List,10 List,10 List,10
## poks.p1    List,18 List,18 List,18 List,18 List,18
## dina.p2    List,10 List,10 List,10 List,10 List,10
## dino.p1    List,15 List,15 List,15 List,15 List,15
## lin.avg.p2 List,9  List,9  List,9  List,9  List,9
## nmf.con.p1 List,13 List,13 List,13 List,13 List,13
## nmf.dis.p2 List,8  List,8  List,8  List,8  List,8
## nmf.com.p1 List,13 List,13 List,13 List,13 List,13
## bkt.p2     List,16 List,16 List,16 List,16 List,16
```

```
dat.1["dino.p1", 3]
```

```
## [[1]]
## Activated information in the context:
##  [1] "students"     "default.vals" "concepts"     "items"
##  [5] "min.ntree"    "max.ntree"    "trans"        "min.depth"
##  [9] "max.depth"    "Q"            "concept.exp"  "M"
## [13] "slip"         "guess"        "dino"
```

```
dat.2 <- gen.apply(ALL.MODELS, list(p1 = p,p2 = p_), multiply = TRUE, n = 5)
dat.2
```

```
##         p1     p2
## exp     List,5 List,5
## irt     List,5 List,5
## poks    List,5 List,5
## dina    List,5 List,5
## dino    List,5 List,5
## lin.avg List,5 List,5
## nmf.con List,5 List,5
## nmf.dis List,5 List,5
## nmf.com List,5 List,5
## bkt     List,5 List,5
```

```
dat.2["nmf.com", "p2"]
```

```
## [[1]]
## [[1]][[1]]
## Activated information in the context:
## [1] "items"        "students"     "default.vals" "M"
## [5] "concepts"     "concept.exp"  "Q"            "nmf.com"
##
## [[1]][[2]]
## Activated information in the context:
## [1] "items"        "students"     "default.vals" "M"
## [5] "concepts"     "concept.exp"  "Q"            "nmf.com"
##
## [[1]][[3]]
## Activated information in the context:
## [1] "items"        "students"     "default.vals" "M"
## [5] "concepts"     "concept.exp"  "Q"            "nmf.com"
##
## [[1]][[4]]
## Activated information in the context:
## [1] "items"        "students"     "default.vals" "M"
## [5] "concepts"     "concept.exp"  "Q"            "nmf.com"
##
## [[1]][[5]]
## Activated information in the context:
## [1] "items"        "students"     "default.vals" "M"
## [5] "concepts"     "concept.exp"  "Q"            "nmf.com"
```

Learning the most probable context from data using `learn`

Let's say we want to get the third data generation from the matching between context `p1` and model `poks`, and use POKS model to learn from this data.

```
poks.data <- dat.2["poks", "p1"][[1]][[3]]
poks.data
```

```
## Activated information in the context:
##  [1] "students"        "default.vals"    "concepts"
##  [4] "items"           "min.ntree"       "max.ntree"
##  [7] "trans"           "min.depth"       "max.depth"
## [10] "density"          "min.it.per.tree" "avg.success"
## [13] "alpha.c"          "alpha.p"         "p.min"
## [16] "max.it.per.tree" "po"               "poks"
```

```
poks.data$poks
```

```
## $R
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    1    1    1    1    1    1    1    1    1     1     1     0     0
##  [2,]    1    1    1    1    1    1    1    1    1     1     1     1     0
##  [3,]    0    0    1    0    1    1    1    0    1     1     1     1     0
##  [4,]    1    1    1    1    1    1    1    1    1     1     1     1     1
##  [5,]    0    1    0    0    0    0    1    1    0     1     1     0     0
##  [6,]    1    0    1    1    1    1    1    1    1     1     1     1     1
##  [7,]    1    1    1    1    0    0    0    1    1     1     1     0     0
##  [8,]    1    0    1    1    1    0    0    1    1     0     1     1     1
##  [9,]    0    0    0    0    1    1    1    0    1     1     1     1     0
## [10,]    0    0    0    0    0    1    0    0    0     1     0     0     0
## [11,]    0    1    0    0    0    1    1    1    0     1     1     0     0
## [12,]    0    0    0    0    0    0    0    0    0     0     1     0     0
## [13,]    0    1    1    0    0    0    0    1    0     1     1     0     0
## [14,]    1    0    0    0    0    0    0    0    1     1     1     0     1
## [15,]    1    0    0    0    1    0    0    1    0     0     1     0     1
## [16,]    1    0    1    1    1    1    1    1    1     1     1     1     1
## [17,]    1    1    1    1    1    1    1    1    1     1     1     1     1
## [18,]    0    0    0    0    0    0    0    0    1     0     1     0     0
## [19,]    1    1    1    1    1    1    1    1    1     1     1     1     1
## [20,]    1    0    0    1    1    1    1    1    1     1     1     1     1
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20]
##  [1,]     1     0     1     1     1     1     0
##  [2,]     1     1     1     1     1     1     1
##  [3,]     1     1     1     1     1     1     1
##  [4,]     1     1     1     1     1     1     1
##  [5,]     1     0     1     1     0     0     0
##  [6,]     1     1     1     1     1     1     1
##  [7,]     0     0     1     0     0     1     0
##  [8,]     0     0     0     1     0     1     1
##  [9,]     1     1     1     0     1     1     0
## [10,]     1     1     0     0     1     0     0
## [11,]     1     0     0     1     0     0     0
## [12,]     0     0     0     0     0     0     0
## [13,]     0     0     1     0     0     0     0
## [14,]     0     0     1     1     0     1     0
## [15,]     0     0     0     0     0     1     0
## [16,]     1     1     1     1     1     1     0
## [17,]     1     1     1     1     1     1     1
## [18,]     0     1     0     0     0     0     0
## [19,]     1     1     1     1     1     1     0
## [20,]     1     1     1     1     1     1     1
##
## $alpha.c
## [1] 0.25
##
## $alpha.p
## [1] 0.25
##
## $p.min
## [1] 0.5
```

```
learn.poks <- learn("poks", data = poks.data$poks)
```

```
## Learning by 'poks' ...
```

```
learn.poks
```

```
## Activated information in the context:
##  [1] "default.vals" "poks"         "student.var"  "avg.success"
##  [5] "state"        "or.t"         "or.f"         "po"
##  [9] "alpha.c"      "alpha.p"      "p.min"        "items"
```

```
learn.poks$po
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    0    1    0    0    0    0    1    0    0     0     0     0     0
##  [2,]    1    0    0    1    0    0    0    0    0     0     0     0     0
##  [3,]    0    1    0    0    0    1    0    0    1     0     0     0     0
##  [4,]    0    1    0    0    0    1    0    0    0     0     0     0     0
##  [5,]    1    0    0    0    0    0    0    1    0     0     1     0     1
##  [6,]    0    0    0    1    0    0    0    0    0     0     0     0     0
##  [7,]    1    0    0    0    0    0    0    0    0     0     0     0     1
##  [8,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [9,]    0    0    1    0    0    0    0    0    0     1     0     0     0
## [10,]    0    0    0    0    0    0    0    0    1     0     0     0     0
## [11,]    1    0    0    0    1    0    0    1    0     0     0     0     1
## [12,]    0    0    0    1    0    0    0    0    0     0     0     0     1
## [13,]    0    0    0    0    1    1    1    0    0     0     1     0     0
## [14,]    0    0    0    0    0    0    1    0    0     0     0     0     0
## [15,]    0    1    1    0    0    0    0    1    0     0     0     0     0
## [16,]    1    0    0    1    0    1    0    0    0     0     0     0     0
## [17,]    0    1    0    1    0    1    0    0    0     0     0     0     0
## [18,]    0    0    0    0    0    0    0    0    0     0     0     1     0
## [19,]    1    0    0    1    0    0    0    0    0     0     0     0     0
## [20,]    0    0    0    1    0    1    0    0    0     0     0     0     0
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20]
##  [1,]     0     0     1     0     0     1     0
##  [2,]     0     0     0     1     0     0     0
##  [3,]     0     0     0     0     0     0     0
##  [4,]     0     0     0     1     0     1     0
##  [5,]     0     0     0     0     0     0     0
##  [6,]     0     0     1     1     0     0     1
##  [7,]     1     0     0     0     0     0     0
##  [8,]     0     1     0     0     0     0     0
##  [9,]     0     0     1     0     0     0     1
## [10,]     0     0     0     0     0     0     0
## [11,]     0     0     0     0     0     0     0
## [12,]     0     1     0     1     1     0     0
## [13,]     0     0     0     0     0     0     0
## [14,]     0     1     0     0     0     0     0
## [15,]     1     0     0     0     0     0     0
## [16,]     0     0     0     1     0     1     1
## [17,]     0     0     0     0     0     1     0
## [18,]     0     0     0     0     0     0     0
## [19,]     0     0     1     1     0     0     0
## [20,]     0     0     1     1     0     0     0
```

If we want to learn from this same data using DINA model, `poks.data$poks` cannot be used because components `p.min`, `alpha.p`, `alpha.c` are meaningless to DINA. In order to do the task, one will need to hand-design this data. DINA requires one additional component besides the response matrix: Q-matrix. Normally, Q-matrix is expected to be expert-defined, however this illustration will just simply generate it randomly.

```
Q <- get.par("Q", p)$value
R <- poks.data$poks$R
dina.data <- list(R=R,Q=Q)
learn.dina <- learn("dina", data = dina.data)
```

```
## Learning by 'dina' ...
```

```
learn.dina
```

```
## Activated information in the context:
##  [1] "default.vals"     "dina"             "Q"
##  [4] "M"                "skill.space"      "skill.dist"
##  [7] "slip"             "guess"            "items"
## [10] "concepts"         "students"         "concept.exp"
## [13] "skill.space.size"
```

Here we have a look at two parameters `skill.space` and `skill.dist`

```
learn.dina$skill.space
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    1    0    0    0    0    1    1    1     1     0     0     0
## [2,]    0    0    1    0    0    0    1    0    0     0     1     1     1
## [3,]    0    0    0    1    0    0    0    1    0     0     1     0     0
## [4,]    0    0    0    0    1    0    0    0    1     0     0     1     0
## [5,]    0    0    0    0    0    1    0    0    0     1     0     0     1
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]     0     0     0     1     1     1     1     1     1     0     0
## [2,]     0     0     0     1     1     1     0     0     0     1     1
## [3,]     1     1     0     1     0     0     1     1     0     1     1
## [4,]     1     0     1     0     1     0     1     0     1     1     0
## [5,]     0     1     1     0     0     1     0     1     1     0     1
##      [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32]
## [1,]     0     0     1     1     1     1     0     1
## [2,]     1     0     1     1     1     0     1     1
## [3,]     0     1     1     1     0     1     1     1
## [4,]     1     1     1     0     1     1     1     1
## [5,]     1     1     0     1     1     1     1     1
```

```
learn.dina$skill.dist
```

```
##  [1] 1.038682e-44 7.486790e-02 1.038682e-44 1.673724e-47 1.483768e-51
##  [6] 1.228654e-51 7.486790e-02 1.049940e-04 8.635937e-09 7.040000e-09
## [11] 1.673724e-47 1.306898e-62 1.228654e-51 2.997507e-53 5.097114e-44
## [16] 4.913437e-39 1.049940e-04 8.985574e-18 1.302381e-15 9.928710e-08
## [21] 2.588213e-01 1.809725e-16 3.126374e-64 1.712231e-47 7.237702e-52
## [26] 4.729649e-02 2.615238e-13 1.473083e-09 5.030112e-02 8.830394e-02
## [31] 5.270359e-02 3.526276e-01
```

Generate synthetic data using `syn`

Generating synthetic data includes three steps:

1. Learn the most probable context from a given data by a specified model.
2. Modify the learned context by keeping some of the parameters, changing some and discarding the rest.
3. Generate data from the new context.

`edmsyn` provides function `syn` that automates the above process, specifically, `syn` consists of three parts:

1. Learn the most probable context by using `learn`.
2. Keep some parameters (the default choice is stored in `edmconst$KEEP`[2]) and discard the rest, also allow the user to change parameter `students`.

```
edmconst$KEEP
```

```
## $exp
## [1] "avg.success" "it.exp"      "student.var"
##
## $irt
## [1] "dis"      "dif"      "abi.mean" "abi.sd"
##
## $poks
## [1] "po"          "student.var" "state"       "alpha.c"     "alpha.p"
## [6] "p.min"
##
## $dina
## [1] "Q"           "skill.space" "skill.dist"  "slip"        "guess"
##
## $dino
## [1] "Q"           "skill.space" "skill.dist"  "slip"        "guess"
##
## $lin.avg
## [1] "Q"         "S.st.var"  "S.con.exp"
##
## $nmf.con
## [1] "Q"           "concept.exp"
```

[2]`edmconst$KEEP` is designed in a way such that, with any new value the user updates for `students`, the new context is still consistent and sufficient, in this sense function `syn` generates synthetic data by creating simulated students.

```
##
## $nmf.dis
## [1] "Q"           "concept.exp"
##
## $nmf.com
## [1] "Q"           "concept.exp"
##
## $bkt
##  [1] "S.st.var"         "S.con.exp"        "L.st.var"
##  [4] "L.con.exp"        "bkt.slip.st.var"  "bkt.slip.it.exp"
##  [7] "bkt.guess.st.var" "bkt.guess.it.exp" "time"
## [10] "order"            "per.item"         "Q"
```

3. Generate synthetic data from this new context by `gen`

Now we synthesize `dina.data` with a new number of student

```
dina.syn <- syn("dina", data = dina.data, students = 12, n = 10)
```

```
## Learning by 'dina' ...
```

```
dina.syn$synthetic[[5]]$dina
```

```
## $R
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
##  [1,]    0    1    1    1    1    1    1    1    1     0     1     1
##  [2,]    0    1    0    1    1    1    1    0    1     1     1     1
##  [3,]    1    1    0    0    0    1    1    1    1     0     1     0
##  [4,]    1    1    1    1    1    1    1    1    1     1     1     1
##  [5,]    0    1    0    0    0    1    1    0    1     0     1     1
##  [6,]    1    1    1    1    1    1    0    1    1     1     1     1
##  [7,]    0    1    1    1    1    1    0    1    1     0     0     0
##  [8,]    1    1    1    1    1    1    0    1    0     1     0     1
##  [9,]    1    0    1    1    0    0    1    1    1     0     1     0
## [10,]    0    0    0    0    0    0    0    0    0     0     1     0
## [11,]    0    0    0    0    0    0    1    0    1     1     1     0
## [12,]    0    0    0    0    0    0    0    0    0     0     0     0
## [13,]    0    0    0    0    0    0    0    0    1     0     1     1
## [14,]    0    0    0    0    1    0    0    0    0     0     0     0
## [15,]    1    1    1    0    0    0    0    1    0     0     0     1
## [16,]    1    0    1    1    1    1    1    1    1     0     1     1
## [17,]    1    1    1    1    1    1    1    1    1     1     1     1
## [18,]    0    0    1    0    0    0    0    0    0     0     1     1
## [19,]    1    1    1    1    1    1    1    1    1     1     1     1
## [20,]    1    1    1    1    1    1    1    1    1     1     1     1
##
## $Q
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    0    0    1    0    0
##  [2,]    1    0    0    1    1
##  [3,]    1    1    0    1    0
##  [4,]    0    0    0    1    1
##  [5,]    1    0    1    1    0
##  [6,]    0    0    1    0    1
##  [7,]    0    0    0    0    1
##  [8,]    0    1    0    1    0
##  [9,]    1    1    1    0    1
## [10,]    1    1    0    0    1
## [11,]    0    1    1    0    1
## [12,]    0    1    0    1    1
## [13,]    0    0    0    1    0
## [14,]    1    1    1    0    1
## [15,]    0    1    1    1    1
## [16,]    1    1    0    0    1
## [17,]    0    0    1    1    0
## [18,]    0    0    0    1    0
## [19,]    1    1    1    1    1
## [20,]    1    0    0    0    0
```

In case the default option is not favoured, `syn` also allows users to manually specify which parameters to keep through argument `keep.pars`.

```
dina.syn <- syn("dina", data = dina.data, keep.pars = c("Q", "concept.exp"), students = 12)
```

```
## Learning by 'dina' ...
```

However, in this case users take their own risk if the kept parameters (together with the new number of students if `students` is redefined), form an inconsistent or insufficient set with respect to the specified model. For example, for synthesizing data by DINA model case, if we choose to keep `M` (which essentially retain the number of students) and define a new number of students, there will be conflict.

```
dina.syn <- syn("dina", data = dina.data, keep.pars = c("Q", "M"), students = 12)
```

```
## Learning by 'dina' ...
```

```
## Error in down.stream(filtered.pars): 'students' receives different values at once
```