Introduction to D3mirt Analysis

`D3MIRT` Modeling

The `D3mirt` analysis is based on descriptive multidimensional item response theory (DMIRT; Reckase, 2009, 1985; Reckase & McKinley, 1991) and can be used to analyze dichotomous and polytomous items to measure latent traits (denoted θ) in a three-dimensional ability space. The method is foremost visual and illustrates item characteristics with the help of vector geometry.

In DMIRT analysis, it is assumed that items in a multidimensional ability space can measure single or multiple latent traits (Reckase, 2009, 1985). This assumption is realized in the *compensatory model*, i.e., a type of measurement model that uses linear combinations of θ-values for ability assessment (Reckase, 2009). DMIRT builds on the results from the former model and seeks to maximize item discrimination in the latent space by relaxing the assumption of unidimensionality. The method is *descriptive* because the results describe both the strength of discrimination of the items and the extent to which the items are unidimensional, i.e., that the items discriminate on one dimension only, or are within-multidimensional, i.e., that the maximum item discrimination is reached when the item measures more than one dimension.

Regarding vector orientation, the angle of the vector arrows indicates what traits, located along the axes in the model, an item measures (Reckase, 2009, 1985; Reckase & McKinley, 1991). For instance, in a two-dimensional space, an item is *unidimensional* if its item vector arrow is at 0ř with respect to one of the axes in the model and 90ř with respect to the other. Such an item describes a single trait only. In contrast, an item is perfectly *within-multidimensional* if its item vector arrow is oriented at 45ř in relation to the axes in the model. Such an item describes both traits in the model equally well. The same criteria are extended to the n-dimensional case.

The DMIRT approach uses two types of item models, dependent on item type. If dichotomous items are used, the analysis is based on the multidimensional two-parameter logistic model (M2PL) (McKinley & Reckase, 1983; Reckase, 1985). If polytomous items are used, the analysis is extended to the multidimensional two-parameter graded response model (MGRM; Muraki & Carlson, 1995). In both cases, the estimation process consists of first fitting a compensatory multidimensional two-parameter model from which loadings (a) and difficulty parameters (d) are extracted. In the case of `D3mirt`, this implies that a compensatory three-dimensional M2PL or MGRM is fitted. In the next step, the algorithm tries to find the angle and distance from the origin of the model indicating the location of the highest possible discrimination value, also referred to as the inflection point, for each item. The location, in turn, shows what latent trait, or traits, the item can be said to measure and on what level of difficulty. The output is visualized as vector arrows representing the domain of the item response functions located in a multidimensional space pointing in the direction of the maximum slope.

The most central estimates, briefly explained below, in DMIRT analysis are the multidimensional discrimination ($MDISC$) parameter (Reckase & McKinley, 1991), the multidimensional difficulty ($MDIFF$) parameter (Reckase, 1985), and the directional discrimination ($DDISC$) parameters (Reckase & McKinley, 1991). The equations for the just-mentioned statistics will be briefly presented below. However, for reasons of simplicity, and without loss of generality, the presentation will be limited to the two-dimensional case and the M2PL.

In DMIRT, it is assumed that axes l are orthogonal. This introduces the constraint that $\sum_{l=1}^{m} cos^2 \alpha_{jl} = 1$, i.e., that squared cosines equal 1 so that $cos^2\, \alpha_{im} = 1 - \sum_{k=1}^{m-1} cos^2\, \alpha_{ik}$, for item i, person j, and k orthogonal axes.

The $MDISC$ for item i represents the highest level of discrimination the item i can achieve located in a multidimensional latent trait space, with m number of dimensions and a_{ik} item slope parameters (Reckase, 2009, Reckase & McKinley, 1991).

$$MDISC := \sqrt{\sum_{k=1}^{m} a_{ik}^2}$$

Similarly to unidimensional discrimination, the $MDISC$ parameter, also denoted as A_i, indicates the steepness of the slope of the item response function at the point of inflection. The slope is, similarly to the unidimensional case, calculated multiplied with the constant $\frac{1}{4}$, which is omitted in the equation above.

The item orientation towards the steepest slope is set by taking the inverse cosine function of the ratio between the a_{il}, i.e., the slope value of item i on the coordinate axis l, and the $MDISC$ (Reckase, 2009, 1985).

$$\omega_{il} = \arccos\left(\frac{a_{il}}{\sqrt{\sum_{k=1}^{m} a_{ik}^2}}\right)$$

The resulting angle ω_{il} is an item characteristic that describes an item's angular orientation in the multidimensional latent trait space.

The multidimensional version of the difficulty parameter, denoted B_i, is the DMIRT counterpart to the b-parameter in the unidimensional models. The $MDIFF$ is defined as the negative intercept d_i over the $MDISC$ (Reckase, 2009, 1985).

$$MDIFF := \frac{-d_i}{\sqrt{\sum_{k=1}^{m} a_{ik}^2}}$$

The $MDIFF$ is interpreted similarly as the difficulty parameter in the unidimensional model. That is, higher values indicate that higher levels of ability for a probability of a correct response of more than 0.5 are necessary. Moreover, the $MDIFF$, just as in the unidimensional model, also indicates the distance from the origin of the model to the point of inflection. However, in DMIRT analysis, the $MDIFF$ becomes a multidimensional location parameter that shows the distance following the item's specific angle orientation.

For Likert items that hold multiple item response functions, the $MDIFF$ will be turned into an index indicating the distance from the origin to the point of inflection for all response functions derived from an item. This, in turn, implies that the $MDIFF$ will become a *difficulty range* for polytomous items in the model.

Lastly, the $MDISC$ is visualized in DMIRT analysis by scaling the length of the vector arrows. In brief, the bottom location coordinates of the vector arrows, given by the $MDIFF$ and direction vector, are multiplied with the $MDISC$ so that items with higher $MDISC$ have longer vector arrows (Reckase, 2009). Accordingly, shorter vector arrows indicate lower discrimination (which suggests model violations in relation to the item model).

Note that because the $MDISC$ represents an item's maximum level of discrimination in the model oriented in a particular direction, the $MDISC$ can not be used to compare item discrimination locally. For the latter to be possible, the orientation of the items must be constrained to follow a specified direction in the model. This can be achieved with the $DDISC$, defined as follows (Reckase & McKinley, 1991):

$$DDISC := \sum_{k=1}^{m} a_{ik} cos\alpha_{ik}$$

The $DDISC$ describes the level of discrimination for one or more items in the model at the angle set by the direction of choice. Note that it is always true that $DDISC \leq MDISC$.

In the `D3mirt` package, the $DDISC$ is used to create *construct vectors*. Constructs, in this context, refer to the assumption that a subset of items or a particular orientation in the model space can measure a higher-order latent variable. This is implemented in `D3mirt` as optional vectors whose orientation is either calculated as the average direction of a subset of items (from all items down to a single item) or indicated by spherical coordinates. A construct vector will, therefore, point in the direction of the maximum slope of what we might think of as an imaginary item response function indicated by the items or coordinates chosen by the user based on exploratory or theoretical reasons.

If constructs are used, the output will include the $DDISC$ parameters, showing how well all the items discriminate under the assumption that they measure in the direction indicated by the constructs in the analysis. Thus, while the $MDISC$ represents the maximum level of discrimination for the items in the model, the $DDISC$ represents local discrimination that can be compared across all items. Constructs are visually represented with vector arrows scaled to an arbitrary length in the output.

Limitations

The DMIRT method is currently limited to using the two-parameter compensatory models with either the 2PL or the GRM as the item model. Regarding `D3mirt`, the number of dimensions in the analysis is limited to three. However, in praxis, the number of dimensions need not necessarily be exactly three but up to three. This is because only two items in the set are required in order to identify the model (see the section on model identification below for more on the statistical requirements). It is sometimes possible to study data containing more than three dimensions if the fourth dimension and up can be nested in the former three.

Overview

The package includes the following main functions.

- `modid()`: D3mirt Model Identification
- `D3mirt()`: 3D DMIRT Model Estimation
- `plot()`: Graphical Output for `D3mirt()`

In what follows, the `D3mirt` procedure and some of its functions and options will be described using the built-in data set "anes0809offwaves". The data set ($N = 1046, M_{age} = 51.33, SD = 14.56, 57\%$ Female) is a subset from the American National Election Survey (ANES) from the 2008-2009 Panel Study Off Wave Questionnaires, December 2009 (DeBell, et al, 2010; https://electionstudies.org/data-center/2008-2009-panel-study/). All items measure moral preferences and are positively scored of Likert type, ranging from 1 = *Strongly Disagree* to 6 = *Strongly Agree*. Demographic variables include age and gender (male/female).

The sections below are sorted under the following headings.

1. Model Identification

- 1.1. The `modid()` Function
  - 1.1.1. Item Selection
  - 1.1.2. The Model Identification Procedure
  - 1.1.3. Trouble Shooting
- 1.2. Criteria For Model Identification

1. Model Identification

1.1 The `modid()` Function

For the `D3mirt` analysis, two items with the following properties must be chosen to identify the compensatory model (Reckase, 2009). The first item should not load on the second and third axes (y and z), while the second item should not load on the third axis (z). If this can be empirically achieved, creating the orthogonal structure necessary for the analysis will be possible. The `modid()` function can help with this step in the process by suggesting what items to use. The function does this based on an algorithmic approach that uses factor and item loading strength to order the model.

1.1.1. Item Selection

If the model is unknown, the factor structure must be explored with exploratory factor analysis (EFA). However, because `D3mirt` analysis is based on the M2PL and the MGRM, it is recommended to use multidimensional item response theory EFA methods, such as the EFA option in `mirt()` (Chalmers, 2012), from the package with the same name, with `itemtype = 'graded'` or `'2PL'`, so that the EFA is performed with the proper item model. The `mirt()` function is integrated into the `modid()` function so that the user only needs to provide a data frame containing empirical data in the function's first argument. However, it is also possible to use the `modid()` function without performing the EFA internally by setting `efa = FALSE` if, for instance, a factor loading data frame is already available.

Note that the EFA is only used to find model identification items that meet the necessary DMIRT model specification requirements. The EFA model itself is discarded after this step in the procedure, and the user can, therefore, try different types of rotation methods and compare model identification results. Because the EFA in the `mirt()` function takes a long time to perform, the item loadings from the EFA for this example were stored in the package file "efa.Rdata" and are loaded in the following code chunk below.

To start, load the data set from the ANES 2008-2009 Panel Study and remove columns for age and gender.

```
# Load Package
library(D3mirt)
# Load data
data("anes0809offwaves")
x <- anes0809offwaves
x <- x[, 3:22] # Remove columns for age and gender
```

If the default mode `efa = TRUE` is not changed, the function performs an EFA with three factors as the default before performing the model identification. If factor loadings are already available, it is possible to assign them to a data frame and use them in the call to `modid()` by setting `efa` to `FALSE`. This allows `modid()` to skip the efa and the model identification will be instant.

```
# Optional: Load the modid data for this example directly from the package file
load(system.file("extdata/id.Rdata", package = "D3mirt"))
```

```
# Call to modid()
id <- modid(x)
```

```
summary(id)
#>
#> modid: 20 items and 3 factors
#>
#> Model identification items:
#> Item 1 W7Q3
#> Item 2 W7Q20
#>
#>       Item.1    ABS
#> W7Q3  0.8547 0.0174
#> W7Q5  0.8199 0.0648
#> W7Q1  0.7589 0.0772
#> W7Q10 0.7239 0.0854
#>
#>       Item.2    ABS
#> W7Q20 0.7723 0.0465
#> W7Q19 0.6436 0.0526
#> W7Q18 0.6777 0.0782
#>
#>    SS Loadings
#> F2      5.3505
#> F1      2.1127
#> F3      1.6744
#>
#>            F2      F1      F3
#> W7Q1   0.7589  0.0407 -0.0365
#> W7Q2   0.8901 -0.0263 -0.0838
```

```
#> W7Q3   0.8547 -0.0096 -0.0078
#> W7Q4   0.6628  0.0272  0.1053
#> W7Q5   0.8199 -0.0390 -0.0258
#> W7Q6   0.6654  0.0525  0.1054
#> W7Q7   0.5604 -0.0148  0.2087
#> W7Q8   0.5731  0.0390  0.1966
#> W7Q9   0.6151  0.0697  0.0918
#> W7Q10  0.7239  0.0371 -0.0483
#> W7Q11  0.2085  0.0959  0.5488
#> W7Q12  0.0755 -0.0853  0.5559
#> W7Q13 -0.0176 -0.0153  0.7654
#> W7Q14 -0.0407  0.1439  0.5629
#> W7Q15  0.1087  0.4556 -0.1111
#> W7Q16  0.1759  0.2100  0.1152
#> W7Q17  0.2160  0.5816  0.0261
#> W7Q18 -0.0560  0.6777 -0.0782
#> W7Q19  0.0589  0.6436  0.0526
#> W7Q20 -0.0735  0.7723  0.0465
```

The output consists of an $S3$ object of class `modid` containing data frames with model identification items, order of factor strength (based on sum of squares), and item factor loadings. Using the `summary()` function, we first get a printed message telling us the number of items and the number of factors used in the analysis together with the suggested model identification items. The items that are suggested by `modid()` to identify the model in this example are the items W7Q3, for the x-axis, and W7Q20, for the y-axis. Next, we find data frames that hold all the model identification items (`Item.1...Item.n`) selected by `modid()` together with the items' absolute sum score (`ABS`), one frame for the sum of squares for factors sorted in descending order, and one frame for item factor loadings.

The absolute sum scores for the model identification items indicate statistical fit to the structural assumptions of the DMIRT model. They are sorted with the lowest absolute sum score highest up. Accordingly, the top items are the items that best meet the necessary statistical requirements for model identification. Note that the order of the factors follows the model identification items so that item 1 comes from the strongest factor, item 2 from the second strongest, and so on.

1.1.2. The Model Identification Procedure

If improper items are chosen, the model will be hard to interpret or even unempirical if the data is forced to fit the model. The `modid()` function was, therefore, developed to hinder these problems and to maximize the interpretive meaning by using an algorithmic approach that can be user-adjusted. In brief, in the default automatic mode, `modid()` starts by first calculating the sum of squares loadings for the number of factors F in the data and then rearranging the columns in x in decreasing order, following the level of strength of the sum of squares loadings. Next, the function creates a list containing the factor loadings on the first factor, f_1, and absolute sum scores of the factor loadings on the remaining factors, i.e., $F - f_1$, row-wise. The list is rearranged in decreasing order row-wise from factor loading strength on f_1.

Next, items are selected by scaling f_1 loadings and then extracting the items with the highest loadings on f_1, up to a standard deviation of 0.5 (the default setting) as the lower bound criteria counting from the top. That is, rows with raw factor scores and absolute sum scores are extracted until the lower bound for factor loadings on f_1 is reached. This allows the function to extract more rows in the case empirical factor loadings are very similar in strength while also excluding weak loading items. As a last step, the function filters the list based on the absolute sum score by using the `upper` bound as the selection criteria, i.e., items with an absolute sum score higher than the `upper` bound are removed. The results are recorded in nested lists before the function starts over with the next factor, f_2, and so on.

Note that the absolute sum score is always assessed based on the number of factors less than the total number of factors, following the order of iteration. That is, iteration 1 uses factor loadings from all factors $F - f_1$, iteration 2 $F - (f_{1,2})$, iteration 3 $F - (f_{1,2,3})$, and so on, when calculating the absolute sum scores.

The model identification procedure orders the entire model so that the strongest loading item, from the strongest factor, always aligns perfectly with the x-axis and that the other items follow thereon. The model identification process is described in more detail below.

1.1.3. Trouble Shooting

Sometimes, however, the model is hard to identify. If this happens, try the following in the suggested order. For more on the function arguments see the section below regarding the model identification procedure.

1. Change the rotation method in the EFA, e.g., change from *oblimin* to *varimax*.
2. Adjust the `lower` bound in `modid()`.
3. Override factor order with `fac.order`.
4. Adjust the `upper` bound in `modid()`.

The latter (point 4) should, however, only be used as a last resort. This is because the upper bound sets the upper limit for item inclusion. Adjusting this limit too high compromises the necessary statistical requirements. The lower limit (point 2), however, only increases the size of the item pool used for the item selection. It is, therefore, recommended to adjust the lower limit up and down to see if the output differs, and from that make the final decision.

```
# Assigns loadings from a to data frame x
x <- id$loadings
```

```
# Call to modid with increased lower and higher bound
# Set efa to false when using factor loadings as data frame
a <- modid(x, efa = FALSE, lower = 1, upper = 1)
summary(a)
#>
#> modid: 20 items and 3 factors
#>
#> Model identification items:
#> Item 1 W7Q3
#> Item 2 W7Q17
#>
#>        Item.1    ABS
#> W7Q3   0.8547 0.0174
#> W7Q5   0.8199 0.0648
#> W7Q1   0.7589 0.0772
#> W7Q10 0.7239 0.0854
#> W7Q2   0.8901 0.1101
#> W7Q4   0.6628 0.1325
#> W7Q6   0.6654 0.1579
#> W7Q9   0.6151 0.1615
#> W7Q7   0.5604 0.2234
#> W7Q8   0.5731 0.2356
#>
#>        Item.2    ABS
#> W7Q17 0.5816 0.0261
#> W7Q20 0.7723 0.0465
```

```
#> W7Q19 0.6436 0.0526
#> W7Q18 0.6777 0.0782
#>
#>    SS Loadings
#> F2      5.3505
#> F1      2.1127
#> F3      1.6744
#>
#>            F2      F1      F3
#> W7Q1   0.7589  0.0407 -0.0365
#> W7Q2   0.8901 -0.0263 -0.0838
#> W7Q3   0.8547 -0.0096 -0.0078
#> W7Q4   0.6628  0.0272  0.1053
#> W7Q5   0.8199 -0.0390 -0.0258
#> W7Q6   0.6654  0.0525  0.1054
#> W7Q7   0.5604 -0.0148  0.2087
#> W7Q8   0.5731  0.0390  0.1966
#> W7Q9   0.6151  0.0697  0.0918
#> W7Q10  0.7239  0.0371 -0.0483
#> W7Q11  0.2085  0.0959  0.5488
#> W7Q12  0.0755 -0.0853  0.5559
#> W7Q13 -0.0176 -0.0153  0.7654
#> W7Q14 -0.0407  0.1439  0.5629
#> W7Q15  0.1087  0.4556 -0.1111
#> W7Q16  0.1759  0.2100  0.1152
#> W7Q17  0.2160  0.5816  0.0261
#> W7Q18 -0.0560  0.6777 -0.0782
#> W7Q19  0.0589  0.6436  0.0526
#> W7Q20 -0.0735  0.7723  0.0465
```

In this case, we find that the second item "W7Q17" is new. Observing the statistical estimates, we can also see that this outcome is related to the increased `lower` bound, allowing weaker loading items to be included in the selection process. Both the previous item ("W7Q20") and the new item ("W7Q17") have an acceptable absolute sum score, however. We can also note that the increased `upper` bound allows more items to be included in the selection process. This is most notable in the first data frame for `Item.1`. In this case, however, this had no effect on the final selection.

Another option (point 3) is to override the factor order with the `fac.order` argument. More specifically, `modid()` orders factor by squared factor loadings so that the strongest factor is used first, the second strongest factor is used second, and so on. Sometimes, however, there is only a very small difference between the squared factor loadings that, in turn, can cause problems, often only observable at later stages. In such situations, it can be helpful to rearrange the factor order manually to see if the model solution improves.

Since the squared factor loadings for factors 2 and 3 in this example are somewhat similar, it could be useful to compare the final results after manually overriding the factor order. The latter is, however, outside of the scope of this vignette.

```
# Override factor order by reversing columns in the original data frame
b <- modid(x, efa = FALSE, fac.order = c(3, 2, 1))
summary(b)
#>
#> modid: 20 items and 3 factors
#>
#> Model identification items:
```

```
#> Item 1 W7Q13
#> Item 2 W7Q18
#>
#>       Item.1    ABS
#> W7Q13 0.7654 0.0329
#>
#>       Item.2    ABS
#> W7Q18 0.6777 0.0560
#> W7Q19 0.6436 0.0589
#> W7Q20 0.7723 0.0735
#>
#>    SS Loadings
#> F3      1.6744
#> F1      2.1127
#> F2      5.3505
#>
#>            F3      F1      F2
#> W7Q1  -0.0365  0.0407  0.7589
#> W7Q2  -0.0838 -0.0263  0.8901
#> W7Q3  -0.0078 -0.0096  0.8547
#> W7Q4   0.1053  0.0272  0.6628
#> W7Q5  -0.0258 -0.0390  0.8199
#> W7Q6   0.1054  0.0525  0.6654
#> W7Q7   0.2087 -0.0148  0.5604
#> W7Q8   0.1966  0.0390  0.5731
#> W7Q9   0.0918  0.0697  0.6151
#> W7Q10 -0.0483  0.0371  0.7239
#> W7Q11  0.5488  0.0959  0.2085
#> W7Q12  0.5559 -0.0853  0.0755
#> W7Q13  0.7654 -0.0153 -0.0176
#> W7Q14  0.5629  0.1439 -0.0407
#> W7Q15 -0.1111  0.4556  0.1087
#> W7Q16  0.1152  0.2100  0.1759
#> W7Q17  0.0261  0.5816  0.2160
#> W7Q18 -0.0782  0.6777 -0.0560
#> W7Q19  0.0526  0.6436  0.0589
#> W7Q20  0.0465  0.7723 -0.0735
```

In this case, we find that item W7Q3, which was previously suggested for the x-axis, is now suggested for the y-axis. For the x-axis, we find a new item, W7Q13.

1.2. Criteria For Model Identification

Model identification items should preferably (a) have an absolute sum score of $\leq .10$ and (b) have the highest factor loading scores on the factor of interest. Of these two criteria, (a) should be given the strongest weight in the selection decision. If these conditions cannot be met, the user is advised to proceed with caution since the loading scores, therefore, imply that an adequate orthogonal structure may not be empirically attainable. If problems in the model identification process occur, please follow the advice given above.

1.3. Limitations

The `modid()` function is not limited to three-dimensional analysis and can be used on any number of dimensions. Although based on suggestions on model identification given by Reckase (2009) for this type of analysis, the function offers some expansions that introduce more precision. The latter foremost consists of incorporating the sum of squares and factor loadings in the item selection process (unless the user has not specified otherwise). Experience tells us that this is a good practice that often leads to better results compared to other known options. However, it is important to recognize that the model identification procedure only gives suggestions to the model specification, and there could be situations where the researcher should consider other methods. Note that two items can be found to identify the model do not imply successful outcomes when using this methodology (i.e., that the model is *good*). But it does suggest that the method *can* be used and the results can be interpreted in a meaningful way.

2. D3mirt model estimation

2.1. The `D3mirt()` Function

The `D3mirt()` function uses two built-in base models fitted with the `mirt` function from the `mirt` package (Chalmers, 2012). The first model is the default model, i.e., the three-dimensional compensatory model, and the latter is the *orthogonal model*, i.e., a restricted model in which the assumption of within-multidimensionality is removed. Note that in `D3mirt` version 2.0.0 and upwards, the choice of model only depends on what type of vector is imputed in the `modid` argument by the user when calling `D3mirt`.

The user also has the option of manually specifying the model fitted with the `mirt()` (Chalmers, 2012) function and then using the resulting S4 object of class 'SingleGroupClass' when calling `D3mirt`. This allows the user to design a unique model for `D3mirt`. To see an example of the model syntax and `mirt` code regarding the default model, see Appendix A.

The `D3mirt()` function returns an $S3$ object of class `D3mirt` with lists of a and d, $MDISC$, and $MDIFF$ parameters, direction cosines, and spherical coordinates. Regarding the latter, spherical coordinates are represented by θ and ϕ. The θ coordinate is the positive or negative angle in degrees, starting from the x-axis, of the vector projections from the vector arrows in the xz-plane up to ± 180ř. Note that the θ angle is oriented following the positive pole of the x and z axis so that the angle increases clockwise in the graphical output. The ϕ coordinate is the positive angle in degrees from the y-axis and the vectors. Note that the ρ coordinate from the spherical coordinate system is in DMIRT represented by the $MDIFF$, and so is reported separately.

If constructs are used, the function also returns construct direction cosines, spherical coordinates for the construct vector arrows, and $DDISC$ parameters (one index per construct).

2.1.1. Calling the Compensatory Model

To use the compensatory model, the user only needs to specify the model identification items from `modid()` in the `modid` argument when calling `D3mirt` as follows. In the example above, `modid()` selected items W7Q3 as item 1 and W7Q20 as item 2. Accordingly, to fit the compensatory model, simply impute a character vector containing the items in the `modid` argument.

Note that because fitting the compensatory model in the `mirt()` function takes a long time, the item parameters for this example are contained in a data frame available in the package file "mod1.Rdata". To save time studying these examples, the package file can be loaded and used in the function call.

```
# Optional: Load mod1 directly from the package file
load(system.file("extdata/mod1.Rdata", package = "D3mirt"))
```

```r
# Call D3mirt() with the model identification items
mod1 <- D3mirt(x, modid = c("W7Q3", "W7Q20"))
```

The `D3mirt()` function prints a short report containing the number of items used and the levels of difficulty of the items when the estimation is done. If construct vectors are used, the function will also print the number of construct vectors and the names of the items included in each construct (see examples below). The `summary()` function is used to investigate the resulting DMIRT estimates.

```r
# Show summary of results
summary(mod1)
#>
#> D3mirt: 20 items and 5 levels of difficulty
#>
#> Compensatory model
#> Model identification items: W7Q3, W7Q20
#>
#>           a1      a2      a3      d1     d2     d3      d4      d5
#> W7Q1  2.0297  0.1645 -0.1227  8.0868 7.0641 5.9876  3.2016 -0.4834
#> W7Q2  2.6215 -0.0025 -0.2576  9.2884 6.6186 4.5103  1.6650 -2.4437
#> W7Q3  2.7932  0.0000  0.0000 10.4884 7.5922 5.6797  2.7181 -1.1790
#> W7Q4  1.9043  0.1877  0.1502  7.3749 6.0464 4.9812  2.4830 -1.1144
#> W7Q5  2.2427 -0.0285 -0.0836  8.4293 6.6722 4.9055  1.8256 -1.8316
#> W7Q6  2.0020  0.2392  0.1578  8.0687 6.3578 4.9521  2.3301 -1.0187
#> W7Q7  1.6284  0.1036  0.3600  6.0180 4.8976 3.6909  1.6326 -1.3483
#> W7Q8  1.7773  0.2254  0.3536  6.9174 5.1824 3.7663  1.4845 -1.8331
#> W7Q9  1.7197  0.2495  0.1286  7.5588 4.9756 3.3649  0.9344 -2.2093
#> W7Q10 1.7697  0.1274 -0.1404  8.3641 5.7399 4.2865  1.9648 -0.6641
#> W7Q11 1.4237  0.4680  1.0449  6.2204 4.6938 3.5443  1.1923 -1.8579
#> W7Q12 0.7601  0.0413  0.9370  4.1361 2.8772 2.3420  1.1791 -0.4240
#> W7Q13 1.1265  0.2914  1.6908  5.8838 4.3950 3.4385  1.8931 -0.6004
#> W7Q14 0.7444  0.4832  0.9787  5.3893 3.9334 3.0259  0.8144 -1.5869
#> W7Q15 0.4551  0.7871 -0.1607  4.3208 3.0545 2.3970  0.9187 -0.9705
#> W7Q16 0.6236  0.4141  0.1798  3.7249 2.0305 1.1658 -0.0612 -1.8084
#> W7Q17 1.1891  1.3414  0.0561  6.9016 5.8026 4.9348  2.7917 -0.0040
#> W7Q18 0.4106  1.3543 -0.1372  3.7838 2.0986 1.4183  0.1829 -1.9855
#> W7Q19 0.8578  1.4100  0.2278  4.4979 2.6484 1.6731  0.3741 -1.9966
#> W7Q20 0.7355  1.9066  0.0000  4.6376 2.3632 1.2791 -0.3430 -2.9188
#>
#>        MDISC  MDIFF1  MDIFF2  MDIFF3  MDIFF4 MDIFF5
#> W7Q1  2.0401 -3.9640 -3.4627 -2.9350 -1.5693 0.2370
#> W7Q2  2.6341 -3.5262 -2.5127 -1.7123 -0.6321 0.9277
#> W7Q3  2.7932 -3.7550 -2.7181 -2.0334 -0.9731 0.4221
#> W7Q4  1.9194 -3.8423 -3.1502 -2.5952 -1.2936 0.5806
#> W7Q5  2.2444 -3.7557 -2.9728 -2.1857 -0.8134 0.8160
#> W7Q6  2.0224 -3.9897 -3.1437 -2.4486 -1.1521 0.5037
#> W7Q7  1.6710 -3.6015 -2.9310 -2.2088 -0.9771 0.8069
#> W7Q8  1.8261 -3.7881 -2.8380 -2.0625 -0.8130 1.0038
#> W7Q9  1.7425 -4.3380 -2.8555 -1.9311 -0.5362 1.2679
#> W7Q10 1.7798 -4.6995 -3.2251 -2.4084 -1.1040 0.3731
#> W7Q11 1.8269 -3.4048 -2.5692 -1.9400 -0.6526 1.0170
#> W7Q12 1.2073 -3.4260 -2.3832 -1.9399 -0.9766 0.3512
#> W7Q13 2.0525 -2.8667 -2.1413 -1.6753 -0.9223 0.2925
#> W7Q14 1.3211 -4.0792 -2.9772 -2.2903 -0.6164 1.2011
```

```
#> W7Q15 0.9232 -4.6800 -3.3085 -2.5963 -0.9951 1.0512
#> W7Q16 0.7699 -4.8384 -2.6375 -1.5143  0.0795 2.3490
#> W7Q17 1.7935 -3.8482 -3.2354 -2.7515 -1.5566 0.0022
#> W7Q18 1.4218 -2.6613 -1.4761 -0.9976 -0.1286 1.3965
#> W7Q19 1.6661 -2.6997 -1.5896 -1.0042 -0.2245 1.1984
#> W7Q20 2.0436 -2.2694 -1.1564 -0.6259  0.1678 1.4283
#>
#>       D.Cos X D.Cos Y D.Cos Z    Theta     Phi
#> W7Q1   0.9949  0.0806 -0.0601  -3.4590 85.3756
#> W7Q2   0.9952 -0.0010 -0.0978  -5.6133 90.0554
#> W7Q3   1.0000  0.0000  0.0000   0.0000 90.0000
#> W7Q4   0.9921  0.0978  0.0783   4.5098 84.3885
#> W7Q5   0.9992 -0.0127 -0.0372  -2.1341 90.7284
#> W7Q6   0.9899  0.1183  0.0780   4.5068 83.2064
#> W7Q7   0.9745  0.0620  0.2154  12.4657 86.4441
#> W7Q8   0.9733  0.1235  0.1936  11.2521 82.9082
#> W7Q9   0.9869  0.1432  0.0738   4.2769 81.7671
#> W7Q10  0.9943  0.0716 -0.0789  -4.5357 85.8965
#> W7Q11  0.7793  0.2561  0.5719  36.2765 75.1588
#> W7Q12  0.6296  0.0342  0.7761  50.9493 88.0399
#> W7Q13  0.5488  0.1420  0.8238  56.3262 81.8384
#> W7Q14  0.5634  0.3657  0.7408  52.7448 68.5472
#> W7Q15  0.4929  0.8525 -0.1740 -19.4451 31.5154
#> W7Q16  0.8101  0.5378  0.2335  16.0800 57.4627
#> W7Q17  0.6630  0.7479  0.0313   2.7031 41.5881
#> W7Q18  0.2888  0.9525 -0.0965 -18.4742 17.7264
#> W7Q19  0.5148  0.8463  0.1367  14.8712 32.1870
#> W7Q20  0.3599  0.9330  0.0000   0.0000 21.0939
```

The summary function displays the factor loadings and the difficulty parameters from the compensatory model. Examples of how DMIRT estimates can be used when reporting results are given below in the item and dimensionality analysis.

2.1.2. Calling the Orthogonal Model

A new addition in version 2.0.0 of `D3mirt` is the option of fitting an orthogonal constrained version of the compensatory model. This model allows the user to restrict items in the model space to only load on one of the model axes. The orthogonal model is foremost useful for studying the items under the assumption that the items are unidimensional and that the unidimensions are uncorrelated. In this context *orthogonal* refers to the geographical interpretation of the graphical output in which all three unidimensions are perpendicular in the model space.

To tell `D3mirt` to use the orthogonal model, create a nested list of items specifying which items load to which the three dimensions. In the example below, items 1 to 10 will be constrained to the x-axis, items 15 to 19 to the y-axis, and items 11 to 14 to the z-axis.

On a side note, using an orthogonal model in `D3mirt` often requires removing poorly fitting items. This is because constraining items already struggling to fit the model will most likely increase the misfit. This will most likely lead to a very large $MDIFF$ range, distorting the graphical output. In the example below, item W7Q16 was removed before fitting the model.

```
# Optional: Load mod2 directly from the package file
load(system.file("extdata/mod2.Rdata", package = "D3mirt"))
```

```r
y <- data.frame(x[,-16])# Remove misfitting item W7Q16
# Call D3mirt() and using the orthogonal model
mod2 <- D3mirt(y, modid = list(c(1:10),   # x-axis
                               c(15:19),  # y-axis
                               c(11:14))) # z-axis
```

```r
summary(mod2)
#>
#> D3mirt: 19 items and 5 levels of difficulty
#>
#> Orthogonal model
#> Item vector 1: W7Q1, W7Q2, W7Q3, W7Q4, W7Q5, W7Q6, W7Q7, W7Q8, W7Q9, W7Q10
#> Item vector 2: W7Q15, W7Q17, W7Q18, W7Q19, W7Q20
#> Item vector 3: W7Q11, W7Q12, W7Q13, W7Q14
#>
#>           a1     a2     a3      d1     d2     d3      d4      d5
#> W7Q1  2.0183 0.0000 0.0000  8.0211 7.0101 5.9451  3.1762 -0.4884
#> W7Q2  2.5545 0.0000 0.0000  9.0536 6.4408 4.3879  1.6178 -2.3976
#> W7Q3  2.7483 0.0000 0.0000 10.3241 7.4782 5.6007  2.6793 -1.1828
#> W7Q4  1.9121 0.0000 0.0000  7.3485 6.0230 4.9697  2.4782 -1.1279
#> W7Q5  2.1971 0.0000 0.0000  8.2981 6.5636 4.8289  1.7951 -1.8163
#> W7Q6  2.0070 0.0000 0.0000  8.0685 6.3532 4.9445  2.3214 -1.0367
#> W7Q7  1.5979 0.0000 0.0000  5.9310 4.8301 3.6411  1.5984 -1.3322
#> W7Q8  1.7491 0.0000 0.0000  6.8381 5.1224 3.7094  1.4480 -1.8096
#> W7Q9  1.7410 0.0000 0.0000  7.5706 4.9930 3.3754  0.9282 -2.2171
#> W7Q10 1.7488 0.0000 0.0000  8.3054 5.6856 4.2405  1.9417 -0.6614
#> W7Q11 0.0000 0.0000 1.5285  5.9525 4.4684 3.3585  1.1019 -1.7346
#> W7Q12 0.0000 0.0000 1.1511  4.1002 2.8570 2.3265  1.1659 -0.4295
#> W7Q13 0.0000 0.0000 1.8866  5.7126 4.2787 3.3450  1.8306 -0.5882
#> W7Q14 0.0000 0.0000 1.3482  5.4784 4.0172 3.1011  0.8363 -1.6128
#> W7Q15 0.0000 0.8870 0.0000  4.3049 3.0375 2.3797  0.9084 -0.9651
#> W7Q17 0.0000 1.5464 0.0000  6.6917 5.6269 4.7725  2.6404 -0.0421
#> W7Q18 0.0000 1.3316 0.0000  3.6937 2.0504 1.3881  0.1841 -1.9387
#> W7Q19 0.0000 1.6471 0.0000  4.5704 2.6646 1.6572  0.3352 -2.0165
#> W7Q20 0.0000 1.9471 0.0000  4.5578 2.3173 1.2468 -0.3427 -2.8570
#>
#>        MDISC  MDIFF1  MDIFF2  MDIFF3  MDIFF4 MDIFF5
#> W7Q1  2.0183 -3.9742 -3.4733 -2.9456 -1.5737 0.2420
#> W7Q2  2.5545 -3.5442 -2.5214 -1.7178 -0.6333 0.9386
#> W7Q3  2.7483 -3.7565 -2.7210 -2.0379 -0.9749 0.4304
#> W7Q4  1.9121 -3.8432 -3.1500 -2.5992 -1.2961 0.5899
#> W7Q5  2.1971 -3.7768 -2.9873 -2.1978 -0.8170 0.8267
#> W7Q6  2.0070 -4.0202 -3.1655 -2.4636 -1.1566 0.5165
#> W7Q7  1.5979 -3.7118 -3.0228 -2.2787 -1.0003 0.8337
#> W7Q8  1.7491 -3.9096 -2.9287 -2.1208 -0.8279 1.0346
#> W7Q9  1.7410 -4.3485 -2.8680 -1.9388 -0.5332 1.2735
#> W7Q10 1.7488 -4.7492 -3.2511 -2.4248 -1.1103 0.3782
#> W7Q11 1.5285 -3.8944 -2.9234 -2.1973 -0.7209 1.1348
#> W7Q12 1.1511 -3.5620 -2.4820 -2.0211 -1.0129 0.3731
#> W7Q13 1.8866 -3.0279 -2.2679 -1.7730 -0.9703 0.3118
#> W7Q14 1.3482 -4.0635 -2.9796 -2.3001 -0.6203 1.1962
#> W7Q15 0.8870 -4.8531 -3.4243 -2.6827 -1.0241 1.0880
#> W7Q17 1.5464 -4.3273 -3.6387 -3.0862 -1.7074 0.0272
```

```
#> W7Q18 1.3316 -2.7738 -1.5398 -1.0424 -0.1383 1.4559
#> W7Q19 1.6471 -2.7749 -1.6178 -1.0062 -0.2035 1.2243
#> W7Q20 1.9471 -2.3408 -1.1902 -0.6403  0.1760 1.4673
#>
#>       D.Cos X D.Cos Y D.Cos Z Theta Phi
#> W7Q1        1       0       0     0  90
#> W7Q2        1       0       0     0  90
#> W7Q3        1       0       0     0  90
#> W7Q4        1       0       0     0  90
#> W7Q5        1       0       0     0  90
#> W7Q6        1       0       0     0  90
#> W7Q7        1       0       0     0  90
#> W7Q8        1       0       0     0  90
#> W7Q9        1       0       0     0  90
#> W7Q10       1       0       0     0  90
#> W7Q11       0       0       1    90  90
#> W7Q12       0       0       1    90  90
#> W7Q13       0       0       1    90  90
#> W7Q14       0       0       1    90  90
#> W7Q15       0       1       0   NaN   0
#> W7Q17       0       1       0   NaN   0
#> W7Q18       0       1       0   NaN   0
#> W7Q19       0       1       0   NaN   0
#> W7Q20       0       1       0   NaN   0
```

As can be seen, all items only load on one of the three dimensions, all direction cosines are, therefore, 1 or 0, and the spherical coordinates are either 0° or 90°. Note that converting to spherical coordinates in the case the item is oriented strictly along the y-axis implies dividing with cos(0) in the arctan function. Because this operation is undefined, it is reported as `NaN` in the output above.

2.1.3. Constructs by Items

Constructs can be included in the analysis in two ways: by creating one or more nested lists indicating what items belong to what construct or by using nested lists with spherical angles indicating the constructs.

Regarding the first option, nested lists can be created containing all items in the set down to a single item. From this, the `D3mirt()` function finds the average direction of the subset of items contained in each nested list by adding and normalizing the direction cosines for the items and scaling the construct direction vector to an arbitrary length. Naturally, if only one item is used to indicate a construct, the construct will be parallel with the direction of that particular item. The length of the construct vector arrows can be adjusted by the user.

This type of construct can contribute to the analysis by visualizing the average direction for a subset set of items and by showing how all items discriminate locally in the direction of the construct vector with the help of the $DDISC$ index.

The constructs included below were grouped based on exploratory reasons, i.e., because these items cluster in the model (observable in the graphical output). To call `D3mirt` with constructs calculated as the average direction of a subset of items, use the `con.items` argument and create nested lists, as the example below illustrates. Note that if a `D3mirt` object already exists, it is possible to extract factor loadings and the difficulty parameters from the compensatory model from the `D3mirt` object and assign them to a new data frame. This new data frame can then be used to refit the object for the purpose of adding constructs, for instance. Following this procedure, the call to `D3mirt` will skip fitting the compensatory model, and the calculation of parameters will be instant. Note that trait scores will not, however, be included in the exported S3 object when using this option.

```r
# Optional: Load mod3 directly from the package file
load(system.file("extdata/mod3.Rdata", package = "D3mirt"))
```

```r
# Call to D3mirt(), including optional nested lists for three constructs
# Item W7Q16 is not included in any construct because of model violations
# The model violations for the W7Q16 item can be seen when plotting the model
con <- list(c(1:10),
            c(11:14),
            c(15:20))
mod3 <- D3mirt(x, modid = c("W7Q3", "W7Q20"), con.items = con)
```

```r
summary(mod3)
#>
#> D3mirt: 20 items and 5 levels of difficulty
#>
#> Compensatory model
#> Model identification items: W7Q3, W7Q20
#>
#> Constructs
#> Item vector 1: W7Q1, W7Q2, W7Q3, W7Q4, W7Q5, W7Q6, W7Q7, W7Q8, W7Q9, W7Q10
#> Item vector 2: W7Q11, W7Q12, W7Q13, W7Q14
#> Item vector 3: W7Q15, W7Q16, W7Q17, W7Q18, W7Q19, W7Q20
#>
#>
#>            a1      a2      a3      d1     d2     d3      d4      d5
#> W7Q1   2.0297  0.1645 -0.1227  8.0868 7.0641 5.9876  3.2016 -0.4834
#> W7Q2   2.6215 -0.0025 -0.2576  9.2884 6.6186 4.5103  1.6650 -2.4437
#> W7Q3   2.7932  0.0000  0.0000 10.4884 7.5922 5.6797  2.7181 -1.1790
#> W7Q4   1.9043  0.1877  0.1502  7.3749 6.0464 4.9812  2.4830 -1.1144
#> W7Q5   2.2427 -0.0285 -0.0836  8.4293 6.6722 4.9055  1.8256 -1.8316
#> W7Q6   2.0020  0.2392  0.1578  8.0687 6.3578 4.9521  2.3301 -1.0187
#> W7Q7   1.6284  0.1036  0.3600  6.0180 4.8976 3.6909  1.6326 -1.3483
#> W7Q8   1.7773  0.2254  0.3536  6.9174 5.1824 3.7663  1.4845 -1.8331
#> W7Q9   1.7197  0.2495  0.1286  7.5588 4.9756 3.3649  0.9344 -2.2093
#> W7Q10 1.7697  0.1274 -0.1404  8.3641 5.7399 4.2865  1.9648 -0.6641
#> W7Q11 1.4237  0.4680  1.0449  6.2204 4.6938 3.5443  1.1923 -1.8579
#> W7Q12 0.7601  0.0413  0.9370  4.1361 2.8772 2.3420  1.1791 -0.4240
#> W7Q13 1.1265  0.2914  1.6908  5.8838 4.3950 3.4385  1.8931 -0.6004
#> W7Q14 0.7444  0.4832  0.9787  5.3893 3.9334 3.0259  0.8144 -1.5869
#> W7Q15 0.4551  0.7871 -0.1607  4.3208 3.0545 2.3970  0.9187 -0.9705
#> W7Q16 0.6236  0.4141  0.1798  3.7249 2.0305 1.1658 -0.0612 -1.8084
#> W7Q17 1.1891  1.3414  0.0561  6.9016 5.8026 4.9348  2.7917 -0.0040
#> W7Q18 0.4106  1.3543 -0.1372  3.7838 2.0986 1.4183  0.1829 -1.9855
#> W7Q19 0.8578  1.4100  0.2278  4.4979 2.6484 1.6731  0.3741 -1.9966
#> W7Q20 0.7355  1.9066  0.0000  4.6376 2.3632 1.2791 -0.3430 -2.9188
#>
#>        MDISC  MDIFF1  MDIFF2  MDIFF3  MDIFF4 MDIFF5
#> W7Q1   2.0401 -3.9640 -3.4627 -2.9350 -1.5693 0.2370
#> W7Q2   2.6341 -3.5262 -2.5127 -1.7123 -0.6321 0.9277
#> W7Q3   2.7932 -3.7550 -2.7181 -2.0334 -0.9731 0.4221
#> W7Q4   1.9194 -3.8423 -3.1502 -2.5952 -1.2936 0.5806
#> W7Q5   2.2444 -3.7557 -2.9728 -2.1857 -0.8134 0.8160
#> W7Q6   2.0224 -3.9897 -3.1437 -2.4486 -1.1521 0.5037
```

```
#> W7Q7  1.6710 -3.6015 -2.9310 -2.2088 -0.9771 0.8069
#> W7Q8  1.8261 -3.7881 -2.8380 -2.0625 -0.8130 1.0038
#> W7Q9  1.7425 -4.3380 -2.8555 -1.9311 -0.5362 1.2679
#> W7Q10 1.7798 -4.6995 -3.2251 -2.4084 -1.1040 0.3731
#> W7Q11 1.8269 -3.4048 -2.5692 -1.9400 -0.6526 1.0170
#> W7Q12 1.2073 -3.4260 -2.3832 -1.9399 -0.9766 0.3512
#> W7Q13 2.0525 -2.8667 -2.1413 -1.6753 -0.9223 0.2925
#> W7Q14 1.3211 -4.0792 -2.9772 -2.2903 -0.6164 1.2011
#> W7Q15 0.9232 -4.6800 -3.3085 -2.5963 -0.9951 1.0512
#> W7Q16 0.7699 -4.8384 -2.6375 -1.5143  0.0795 2.3490
#> W7Q17 1.7935 -3.8482 -3.2354 -2.7515 -1.5566 0.0022
#> W7Q18 1.4218 -2.6613 -1.4761 -0.9976 -0.1286 1.3965
#> W7Q19 1.6661 -2.6997 -1.5896 -1.0042 -0.2245 1.1984
#> W7Q20 2.0436 -2.2694 -1.1564 -0.6259  0.1678 1.4283
#>
#>       D.Cos X D.Cos Y D.Cos Z    Theta     Phi
#> W7Q1   0.9949  0.0806 -0.0601  -3.4590 85.3756
#> W7Q2   0.9952 -0.0010 -0.0978  -5.6133 90.0554
#> W7Q3   1.0000  0.0000  0.0000   0.0000 90.0000
#> W7Q4   0.9921  0.0978  0.0783   4.5098 84.3885
#> W7Q5   0.9992 -0.0127 -0.0372  -2.1341 90.7284
#> W7Q6   0.9899  0.1183  0.0780   4.5068 83.2064
#> W7Q7   0.9745  0.0620  0.2154  12.4657 86.4441
#> W7Q8   0.9733  0.1235  0.1936  11.2521 82.9082
#> W7Q9   0.9869  0.1432  0.0738   4.2769 81.7671
#> W7Q10  0.9943  0.0716 -0.0789  -4.5357 85.8965
#> W7Q11  0.7793  0.2561  0.5719  36.2765 75.1588
#> W7Q12  0.6296  0.0342  0.7761  50.9493 88.0399
#> W7Q13  0.5488  0.1420  0.8238  56.3262 81.8384
#> W7Q14  0.5634  0.3657  0.7408  52.7448 68.5472
#> W7Q15  0.4929  0.8525 -0.1740 -19.4451 31.5154
#> W7Q16  0.8101  0.5378  0.2335  16.0800 57.4627
#> W7Q17  0.6630  0.7479  0.0313   2.7031 41.5881
#> W7Q18  0.2888  0.9525 -0.0965 -18.4742 17.7264
#> W7Q19  0.5148  0.8463  0.1367  14.8712 32.1870
#> W7Q20  0.3599  0.9330  0.0000   0.0000 21.0939
#>
#>    C.Cos X C.Cos Y C.Cos Z   Theta     Phi
#> C1  0.9970  0.0688  0.0368  2.1119 86.0548
#> C2  0.6409  0.2029  0.7404 49.1207 78.2961
#> C3  0.5405  0.8411  0.0226  2.3974 32.7476
#>
#>       DDISC1 DDISC2 DDISC3
#> W7Q1  2.0304 1.2433 1.2326
#> W7Q2  2.6038 1.4887 1.4088
#> W7Q3  2.7847 1.7901 1.5096
#> W7Q4  1.9169 1.3697 1.1905
#> W7Q5  2.2308 1.3696 1.1862
#> W7Q6  2.0181 1.4484 1.2868
#> W7Q7  1.6438 1.3312 0.9754
#> W7Q8  1.8004 1.4465 1.1582
#> W7Q9  1.7364 1.2479 1.1422
#> W7Q10 1.7679 1.0560 1.0604
```

```
#> W7Q11 1.4900 1.7809 1.1867
#> W7Q12 0.7951 1.1892 0.4668
#> W7Q13 1.2053 2.0328 0.8922
#> W7Q14 0.8113 1.2997 0.8308
#> W7Q15 0.5019 0.3324 0.9043
#> W7Q16 0.6568 0.6168 0.6894
#> W7Q17 1.2799 1.0758 1.7722
#> W7Q18 0.4975 0.4363 1.3578
#> W7Q19 0.9605 1.0044 1.6547
#> W7Q20 0.8644 0.8581 2.0011
```

In contrast to the previous example, the printed message now includes the construct vectors and their respective items. Moreover, compared to the previous function call to `D3mirt()`, there are also two extra data frames: one frame with direction cosines ($D.CosX, D.CosY, D.CosZ$) and spherical coordinates (θ, ϕ) for the constructs, and one frame with the $DDISC$ estimates, containing the $DDISC_{1,2,3}$ parameters for the items corresponding to the three constructs $(1, 2, 3)$ (assigned with c in the example above).

2.1.4. Constructs by Spherical Angles

Regarding the second option, adding a construct arbitrarily in the model is also possible using spherical angles. In this case, the user follows the same convention presented above, i.e., θ to indicate the angle in the xz-plane and ϕ to indicate the angle away from the y-axis. The resulting vector will then be oriented in the direction of the spherical coordinates and report the $DDISC$ for all items in the set constrained to this orientation. In the example below, the constructs are oriented 45° between each axes in the positive direction. More specifically, this implies that the first construct is oriented 45° between the x and y-axes, the second construct 45° between the x and z-axes and the third construct 45° between the y and z-axes.

```
# Optional: Load mod3 directly from the package file
load(system.file("extdata/mod4.Rdata", package = "D3mirt"))
```

```
# Call to D3mirt(), including optional nested lists with spherical angles
# Item W7Q16 is not included in any construct because of model violations
con <- list(c(0, 45),   # theta = 0°, phi = 45°
            c(45, 90),  # theta = 45°, phi = 90°
            c(90, 45))  # theta = 90°, phi = 45°
mod4 <- D3mirt(x, modid = c("W7Q3", "W7Q20"), con.sphe = con)
```

```
summary(mod4)
#>
#> D3mirt: 20 items and 5 levels of difficulty
#>
#> Compensatory model
#> Model identification items: W7Q3, W7Q20
#>
#> Constructs
#> Spherical coordinate vector 1: 0, 45
#> Spherical coordinate vector 2: 45, 90
#> Spherical coordinate vector 3: 90, 45
#>
#>           a1      a2      a3      d1     d2     d3      d4      d5
#> W7Q1  2.0297  0.1645 -0.1227  8.0868 7.0641 5.9876  3.2016 -0.4834
#> W7Q2  2.6215 -0.0025 -0.2576  9.2884 6.6186 4.5103  1.6650 -2.4437
```

```
#> W7Q3  2.7932  0.0000  0.0000 10.4884 7.5922 5.6797  2.7181 -1.1790
#> W7Q4  1.9043  0.1877  0.1502  7.3749 6.0464 4.9812  2.4830 -1.1144
#> W7Q5  2.2427 -0.0285 -0.0836  8.4293 6.6722 4.9055  1.8256 -1.8316
#> W7Q6  2.0020  0.2392  0.1578  8.0687 6.3578 4.9521  2.3301 -1.0187
#> W7Q7  1.6284  0.1036  0.3600  6.0180 4.8976 3.6909  1.6326 -1.3483
#> W7Q8  1.7773  0.2254  0.3536  6.9174 5.1824 3.7663  1.4845 -1.8331
#> W7Q9  1.7197  0.2495  0.1286  7.5588 4.9756 3.3649  0.9344 -2.2093
#> W7Q10 1.7697  0.1274 -0.1404  8.3641 5.7399 4.2865  1.9648 -0.6641
#> W7Q11 1.4237  0.4680  1.0449  6.2204 4.6938 3.5443  1.1923 -1.8579
#> W7Q12 0.7601  0.0413  0.9370  4.1361 2.8772 2.3420  1.1791 -0.4240
#> W7Q13 1.1265  0.2914  1.6908  5.8838 4.3950 3.4385  1.8931 -0.6004
#> W7Q14 0.7444  0.4832  0.9787  5.3893 3.9334 3.0259  0.8144 -1.5869
#> W7Q15 0.4551  0.7871 -0.1607  4.3208 3.0545 2.3970  0.9187 -0.9705
#> W7Q16 0.6236  0.4141  0.1798  3.7249 2.0305 1.1658 -0.0612 -1.8084
#> W7Q17 1.1891  1.3414  0.0561  6.9016 5.8026 4.9348  2.7917 -0.0040
#> W7Q18 0.4106  1.3543 -0.1372  3.7838 2.0986 1.4183  0.1829 -1.9855
#> W7Q19 0.8578  1.4100  0.2278  4.4979 2.6484 1.6731  0.3741 -1.9966
#> W7Q20 0.7355  1.9066  0.0000  4.6376 2.3632 1.2791 -0.3430 -2.9188
#>
#>        MDISC  MDIFF1  MDIFF2  MDIFF3  MDIFF4 MDIFF5
#> W7Q1  2.0401 -3.9640 -3.4627 -2.9350 -1.5693 0.2370
#> W7Q2  2.6341 -3.5262 -2.5127 -1.7123 -0.6321 0.9277
#> W7Q3  2.7932 -3.7550 -2.7181 -2.0334 -0.9731 0.4221
#> W7Q4  1.9194 -3.8423 -3.1502 -2.5952 -1.2936 0.5806
#> W7Q5  2.2444 -3.7557 -2.9728 -2.1857 -0.8134 0.8160
#> W7Q6  2.0224 -3.9897 -3.1437 -2.4486 -1.1521 0.5037
#> W7Q7  1.6710 -3.6015 -2.9310 -2.2088 -0.9771 0.8069
#> W7Q8  1.8261 -3.7881 -2.8380 -2.0625 -0.8130 1.0038
#> W7Q9  1.7425 -4.3380 -2.8555 -1.9311 -0.5362 1.2679
#> W7Q10 1.7798 -4.6995 -3.2251 -2.4084 -1.1040 0.3731
#> W7Q11 1.8269 -3.4048 -2.5692 -1.9400 -0.6526 1.0170
#> W7Q12 1.2073 -3.4260 -2.3832 -1.9399 -0.9766 0.3512
#> W7Q13 2.0525 -2.8667 -2.1413 -1.6753 -0.9223 0.2925
#> W7Q14 1.3211 -4.0792 -2.9772 -2.2903 -0.6164 1.2011
#> W7Q15 0.9232 -4.6800 -3.3085 -2.5963 -0.9951 1.0512
#> W7Q16 0.7699 -4.8384 -2.6375 -1.5143  0.0795 2.3490
#> W7Q17 1.7935 -3.8482 -3.2354 -2.7515 -1.5566 0.0022
#> W7Q18 1.4218 -2.6613 -1.4761 -0.9976 -0.1286 1.3965
#> W7Q19 1.6661 -2.6997 -1.5896 -1.0042 -0.2245 1.1984
#> W7Q20 2.0436 -2.2694 -1.1564 -0.6259  0.1678 1.4283
#>
#>       D.Cos X D.Cos Y D.Cos Z    Theta     Phi
#> W7Q1   0.9949  0.0806 -0.0601  -3.4590 85.3756
#> W7Q2   0.9952 -0.0010 -0.0978  -5.6133 90.0554
#> W7Q3   1.0000  0.0000  0.0000   0.0000 90.0000
#> W7Q4   0.9921  0.0978  0.0783   4.5098 84.3885
#> W7Q5   0.9992 -0.0127 -0.0372  -2.1341 90.7284
#> W7Q6   0.9899  0.1183  0.0780   4.5068 83.2064
#> W7Q7   0.9745  0.0620  0.2154  12.4657 86.4441
#> W7Q8   0.9733  0.1235  0.1936  11.2521 82.9082
#> W7Q9   0.9869  0.1432  0.0738   4.2769 81.7671
#> W7Q10  0.9943  0.0716 -0.0789  -4.5357 85.8965
#> W7Q11  0.7793  0.2561  0.5719  36.2765 75.1588
```

```
#> W7Q12  0.6296  0.0342  0.7761  50.9493 88.0399
#> W7Q13  0.5488  0.1420  0.8238  56.3262 81.8384
#> W7Q14  0.5634  0.3657  0.7408  52.7448 68.5472
#> W7Q15  0.4929  0.8525 -0.1740 -19.4451 31.5154
#> W7Q16  0.8101  0.5378  0.2335  16.0800 57.4627
#> W7Q17  0.6630  0.7479  0.0313   2.7031 41.5881
#> W7Q18  0.2888  0.9525 -0.0965 -18.4742 17.7264
#> W7Q19  0.5148  0.8463  0.1367  14.8712 32.1870
#> W7Q20  0.3599  0.9330  0.0000   0.0000 21.0939
#>
#>    C.Cos X C.Cos Y C.Cos Z Theta Phi
#> C1  0.7071  0.7071  0.0000     0  45
#> C2  0.7071  0.0000  0.7071    45  90
#> C3  0.0000  0.7071  0.7071    90  45
#>
#>       DDISC1 DDISC2  DDISC3
#> W7Q1  1.5515 1.3485  0.0296
#> W7Q2  1.8519 1.6715 -0.1840
#> W7Q3  1.9751 1.9751  0.0000
#> W7Q4  1.4792 1.4527  0.2389
#> W7Q5  1.5656 1.5267 -0.0793
#> W7Q6  1.5848 1.5272  0.2807
#> W7Q7  1.2248 1.4060  0.3278
#> W7Q8  1.4161 1.5067  0.4094
#> W7Q9  1.3925 1.3070  0.2674
#> W7Q10 1.3414 1.1521 -0.0092
#> W7Q11 1.3376 1.7456  1.0698
#> W7Q12 0.5667 1.2001  0.6918
#> W7Q13 1.0026 1.9921  1.4016
#> W7Q14 0.8680 1.2184  1.0337
#> W7Q15 0.8783 0.2082  0.4429
#> W7Q16 0.7338 0.5681  0.4199
#> W7Q17 1.7894 0.8805  0.9882
#> W7Q18 1.2479 0.1933  0.8606
#> W7Q19 1.6036 0.7676  1.1581
#> W7Q20 1.8682 0.5201  1.3482
```

For instance, this type of construct can contribute to the analysis by showing how the items discriminate according to the *DDISC* index in between the axes in model where no item is located. Thus, the example with the 45° construct vectors above can, therefore, be used to study the items assuming within-multidimensionality.

3. Plotting

3.1 The `plot()` Function

The `plot()` method for class `D3mirt` objects is built on functions from the `rgl` package (Adler & Murdoch, 2023) for visualization with OpenGL. The output consists of a three-dimensional interactive RGL device, displaying vector arrows with the latent dimensions running along the orthogonal axes centered at zero. If polytomous items are used, each item will have multiple arrows, representing the multiple item step response functions, running along the same direction in the model.

When plotting the `D3mirt` model with `plot()`, it is possible to visually observe statistical violations in the graphical output returned. For instance, shorter vector arrows indicate weaker discrimination and, therefore, also higher amounts of model violations. As another example, if an item struggles or even fails to describe any of the latent variables in the model, it can often lead to an extreme stretch of the $MDIFF$ range. This is comparable to trace lines turning horizontal in a unidimensional item response theory model. Some examples of model violations and within-dimensionality will be given in the illustration below.

Graphing in default mode by calling `plot()` will return an RGL device that will appear in an external window as a three-dimensional interactive object that can be rotated manually by the user. In this illustration, however, all RGL devices are plotted as still shots.

```
# Plot RGL device
# Graphical output can be seen in Figure 1
plot(mod1, scale = TRUE)
```

An example of how the output can be described could be the following.

> As seen in Figure 1, the pattern in the data indicates the presence of foremost two principal latent constructs indicated by the items, one aligned with the x-axis and one approaching the y-axis. We might also suspect the presence of a third construct located close to the xy-plane, between the x and z axes. Studying the content of the items, the labels *Compassion*, *Fairness*, and *Conformity* were introduced.

We can also study the orthogonal model by calling `plot()` and changing the first argument accordingly. This type of model is, however, less interesting for the purpose of studying item vectors because these simply form a cross pattern in the model space. The model can, however, be interesting plot when studying respondent trait scorers, which will be illustrated below. In the function call, all axes are also named following the content of the items using the `x.lab`, `y.lab`, and `z.lab` arguments.

```
# Plot RGL device
# Graphical output can be seen in Figure 2
plot(mod2, x.lab = "Compassion", y.lab = "Conformity", z.lab = "Fairness")
```

It can be observed that the discrimination scores are lower for all items in the orthogonal model compared to the compensatory model. It is also notable that the size of the reduction is related to how much the item orientation changes when the items are constrained to be parallel with the axes. Accordingly, the discrimination scores for the Compassion items are less affected on average compared to the Fairness items and many of the Conformity items.

3.1.1. Plotting Constructs

To get a better understanding of the interrelation of the data, constructs can be added to asses the average angle when grouping items. As mentioned above, in this example, the items have been grouped in constructs in an exploratory fashion, i.e., based on their observed location in the model. Note that item W7Q16 was not included in any of the constructs because of a high amount of model violation tendencies and notable within-multidimensionality.

To plot constructs alongside item vectors, change to `constructs = TRUE`. The constructs can be named with strings as input to the `construct.lab` argument. In this example, construct labels were chosen based on item content.

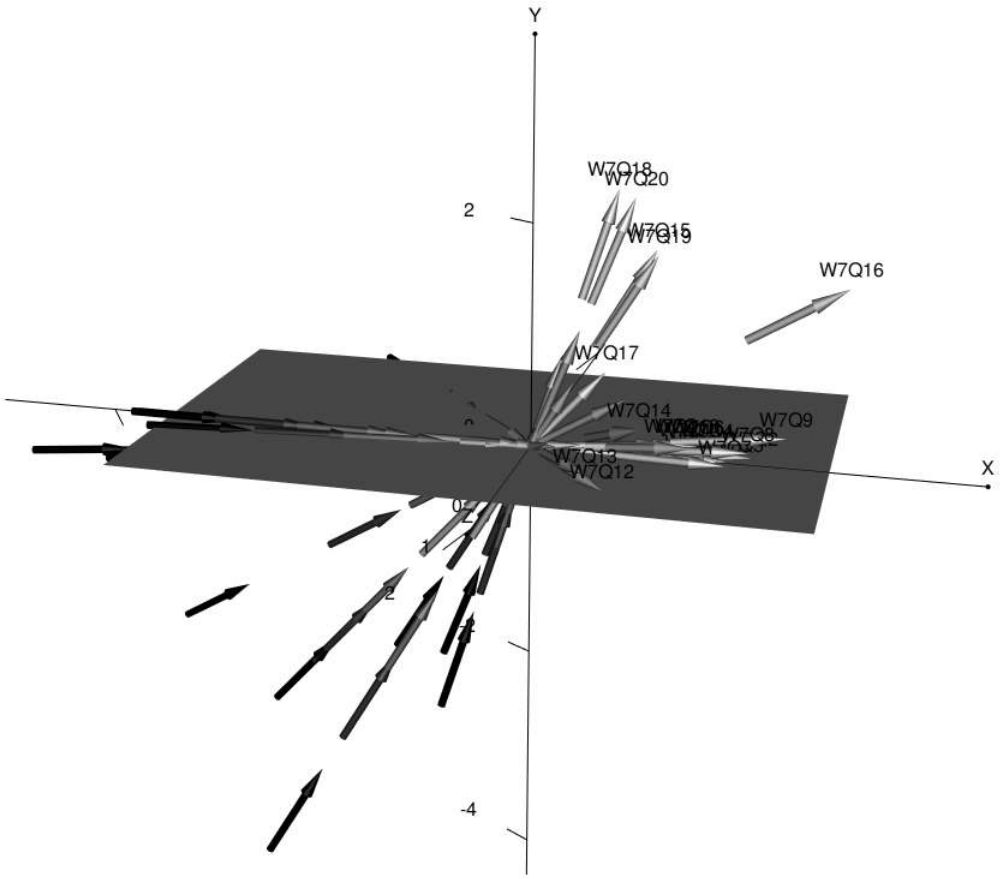

Figure 1: Compensatory Model 1 illustrating multidimensional item charachteristics rotated 15° clockwise.

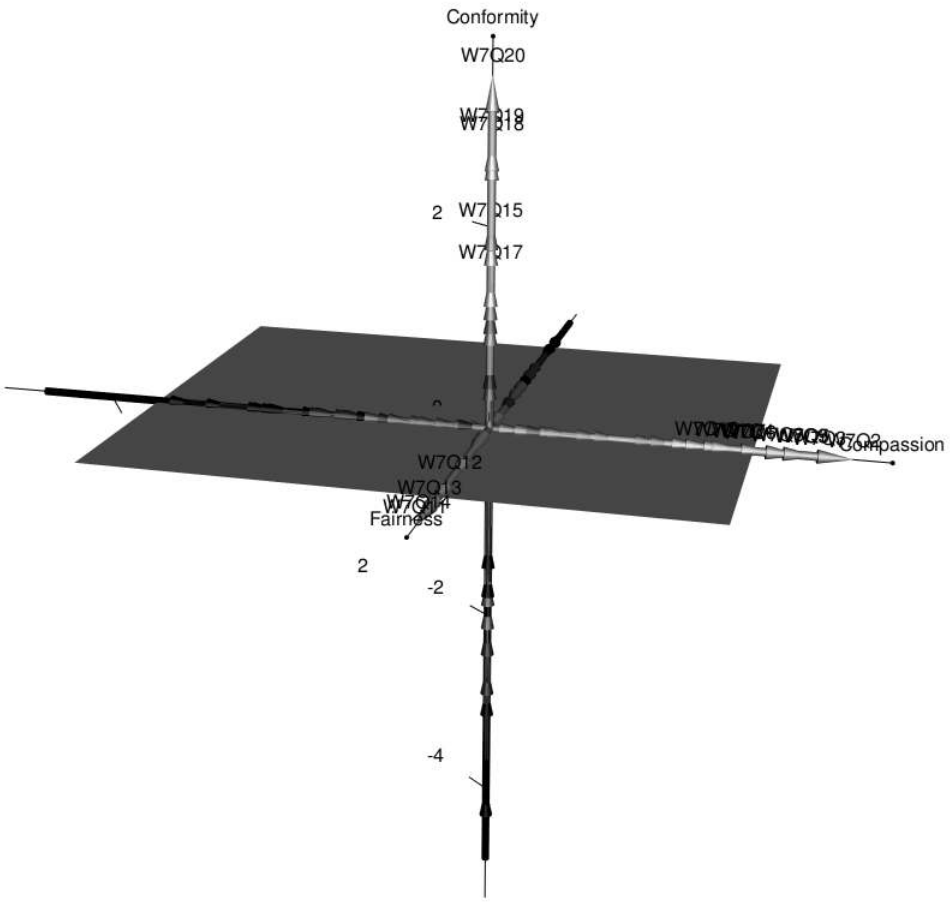

Figure 2: The Orthogonal, Model 2, with item vectors strictly paralell with the model axes, rotated 15° clockwise.

```
# Plot RGL device with constructs visible and named
# Graphical output can be seen in Figure 3
plot(mod3, constructs = TRUE, construct.lab = c("Compassion", "Fairness", "Conformity"))
```

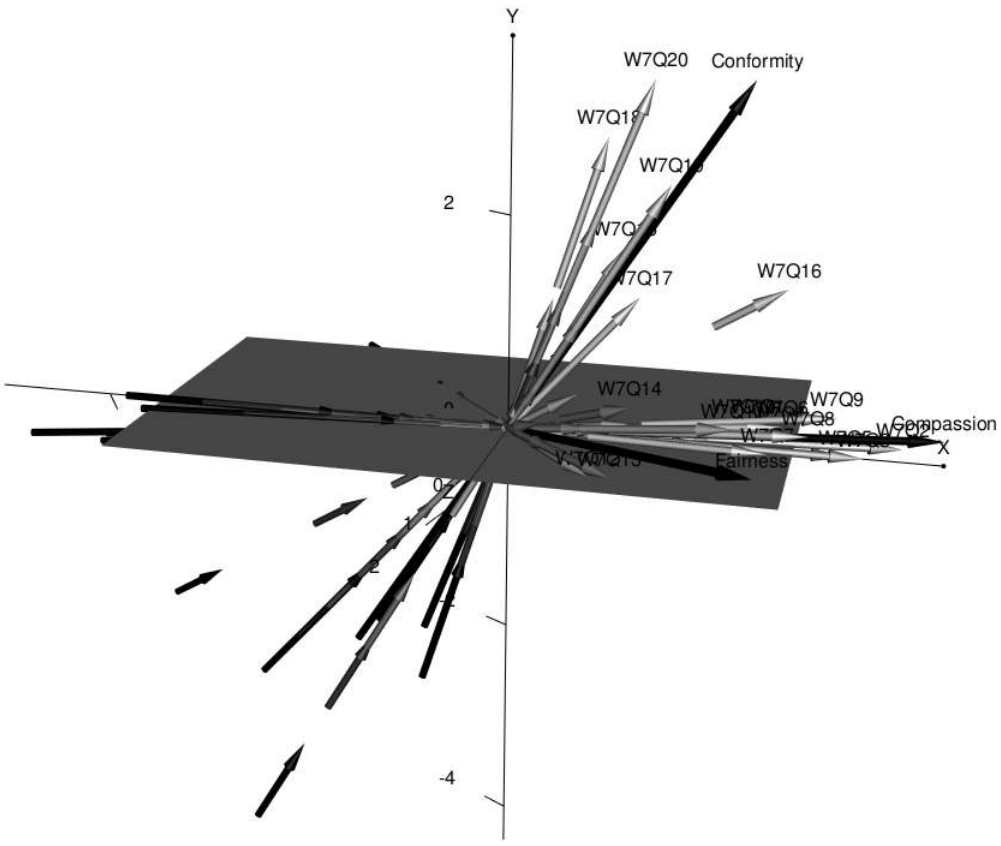

Figure 3: Model 3 rotated 15° showing constructs (solid black arrows) based on item subsets.

An example of how the output can be described is as follows.

> The angles of the constructs inform us that Compassion ($\theta = 2.092°$, $\phi = 86.061°$) and Conformity ($\theta = -2.514°$,$\phi = 28.193°$) have some within-multidimensional tendencies (Figure 3). Still they are both more or less orthogonal to the z-axis. Next, we find Fairness ($\theta = 49.101°$, $\phi = 78.313°$) with clear within-multidimensional tendencies with respect to the x-axis. Thus, the

output indicates that Compassion and Conformity could be independent constructs but Fairness seems not to be.

We can also study the construct indicated by spherical angles by calling `plot()` and changing the first argument to use the appropriate `D3mirt` object. Because these constructs are not empirical, however, the name argument is changed to use the more generic labels "Con 1", "Con 2", and "Con 3".

```
# Plot RGL device with constructs visible
# Graphical output can be seen in Figure 4
plot(mod4, constructs = TRUE, construct.lab = c("Con 1", "Con 2", "Con 3"))
```

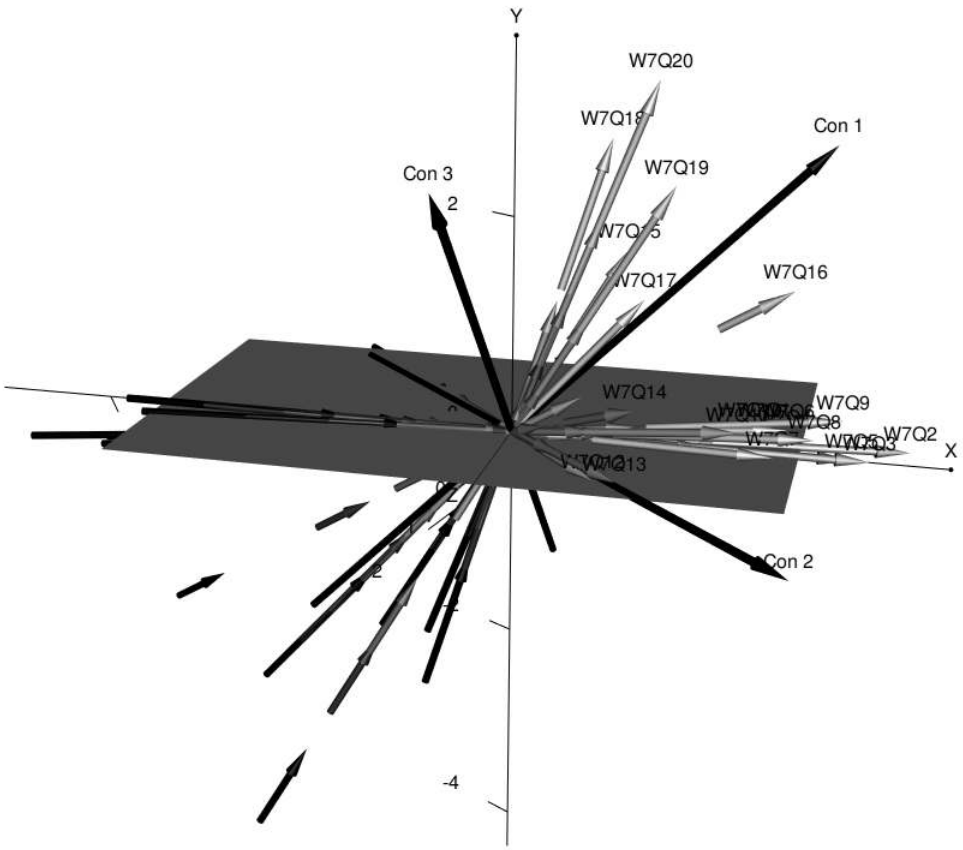

Figure 4: Model 4 rotated 15° showing constructs (solid black arrows) based on spherical coordinates.

Studying the $DDISC$ scores for the constructs in the summary output above, it can be confirmed that Con 1 forms a within-multidimensional scale with the highest discrimination scores in items from Compassion and Conformity, Con 2 describes a within-multidimensional scale with the highest discrimination scores in items from Compassion and Fairness. Con 3, on the other hand, does not seem to result in a scale since the discrimination scores are low and even approaching zero on many items.

3.1.2. `items`

A subset of items can be plotted for a more thorough investigation using the `items` argument. In the example below, all constructs are plotted together with the items used for the conformity construct. In the function call, the numerical indicators in the `items` argument follow the item order in the original data frame (see `?anes0809offwaves`).

```
# A selection of Conformity items from the model plotted with constructs
# Graphical output can be seen in Figure 5
plot(mod3, constructs = TRUE, items = c(15, 17, 18, 19, 20),
     construct.lab = c("Compassion", "Fairness", "Conformity"))
```

The `plot()` function also allows plotting a single item by entering a number indicating what item should be displayed. As was mentioned above, the W7Q16 was not included in any of the constructs because the item showed signs of measurement problems. For example, the short vector arrows indicate high amounts of model violations and the location of the item in the model also indicates that the item is within-multidimensional and that it does not seem to belong to any construct explicitly. Typing 16 in the `items` argument allows for a closer look.

```
# Item W7Q16 has location 16 in the data set (gender and age excluded)
# The item is plotted together with constructs to aid the visual interpretation
# Graphical output can be seen in Figure 6
plot(mod3, constructs = TRUE, items = 16,
     construct.lab = c("Compassion", "Fairness", "Conformity"))
```

An example of how the output can be described is as follows.

> Figure 6 shows that item W7Q16 is located at $\theta = 16.085°$, $\phi = 57.476°$, indicating that the item is within-multidimensional with respect to the x and y-axis; but much less so with respect to the z-axis. In addition, the directional discrimination further underscores that the item does not seem to measure any particular construct ($DDISC_1 = .657$, $DDISC_2 = .617$, $DDISC_3 = .656$). The global discrimination ($MDISC = .770$, $MDIFF_{range} = [-4.838, 2.349]$) is also the lowest of all discrimination scores in the model. This, combined, implies that the item in question does not seem to fit the three-dimensional DMIRT used in this analysis and should, therefore, be removed or adapted. On a side note, we can also note that item W7Q15, $MDISC = .923$, $MDIFF_{range} = [-4.680, 1.051]$) has the second lowest global discrimination score. However, this item does seem to belong to the Conformity construct, observable when comparing angle orientation ($\theta = -19.432°, \phi = 31.515°$) and directional discrimination ($DDISC_1 = .502$, $DDISC_2 = .332$, $DDISC_3 = .912$).

3.1.3. `diff.level`

The user has the option of plotting one level of difficulty at a time with the `diff.level` argument studying the entire scale, a subset of items, or one item at a time. Note that *difficulty* refers to the number of item response functions in the items, i.e., the total number of response options minus one. In this case, 6 response options were used, which means that the model has 5 levels of difficulty.

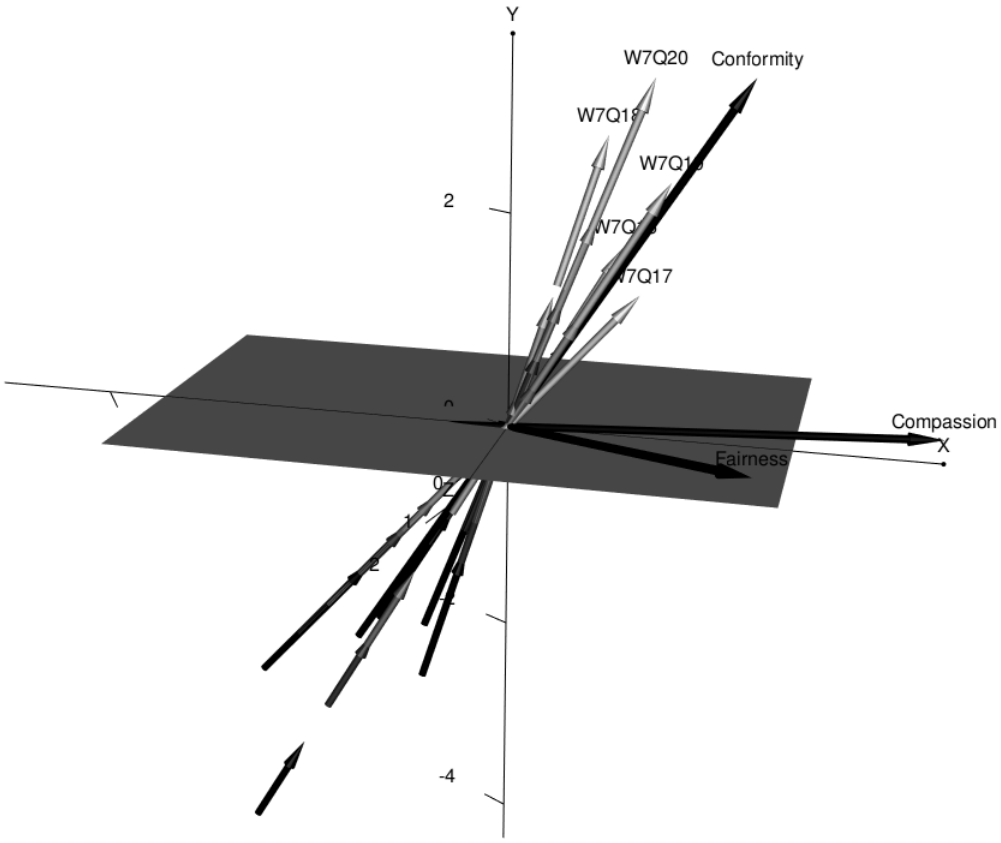

Figure 5: Model 3 rotated 15° showing items 15 to 20 and the three item subset constructs.

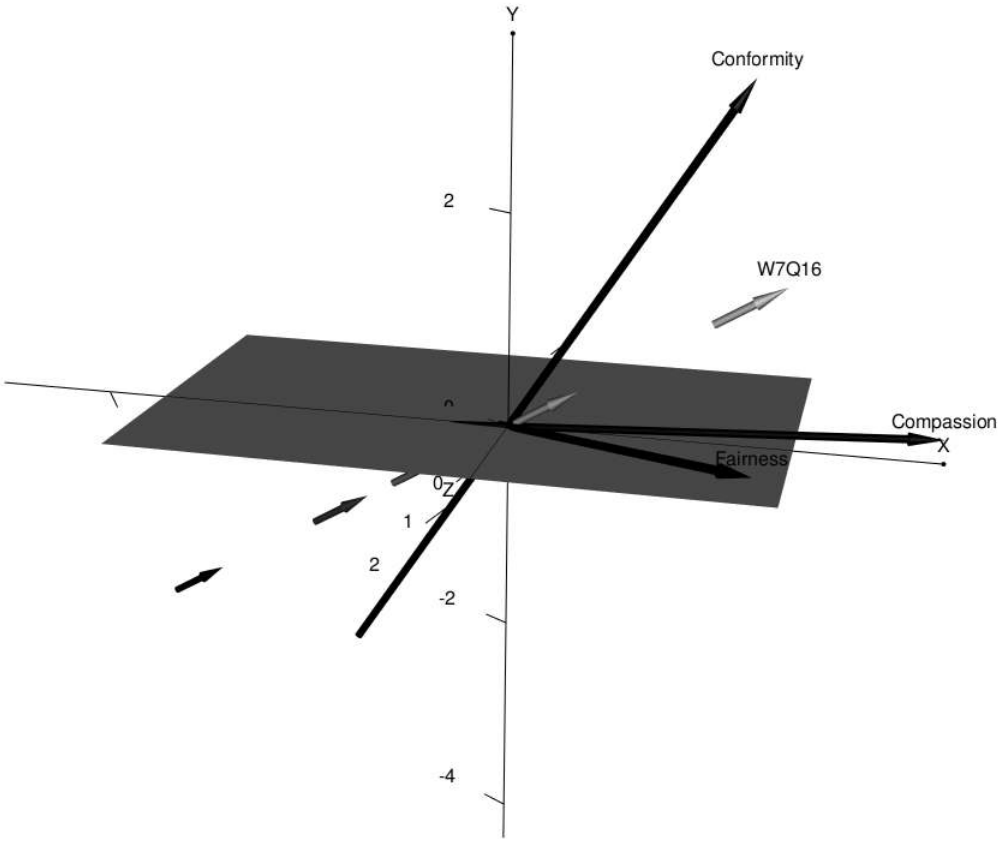

Figure 6: Model 3 rotated 15° showing items 15 and the three item subset constructs.

```
# Plot RGL device on item difficulty level 5
# Graphical output can be seen in Figure 7
plot(mod3, diff.level = 5)
```

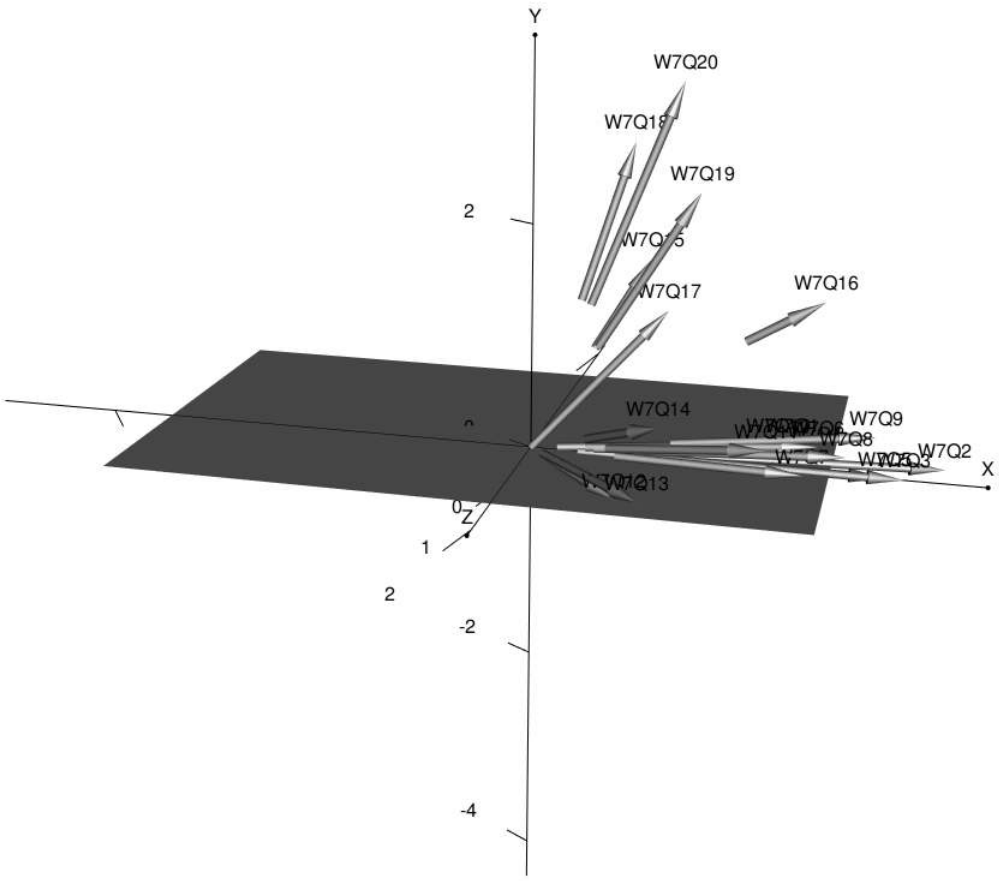

Figure 7: Model 3 rotated 15° clockwise showing all items on difficulty level 5.

3.1.4. `scale`

The `D3mirt()` function returns item vector coordinates estimated with and without the $MDISC$ as a scalar for the arrow length. When the $MDISC$ is not used for the arrow length, all item vector arrows are scaled to one unit length. This allows the user to graph the item vector arrows with `plot()` set to a uniform length.

This can help reduce visual clutter in the graphical output. To view the item vector arrows without the $MDISC$, set `scale = TRUE`.

```
# Plot RGL device with items in uniform length and constructs visible and named
# Graphical output can be seen in Figure 8
plot(mod3, scale = TRUE, constructs = TRUE,
     construct.lab = c("Compassion", "Fairness", "Conformity"))
```

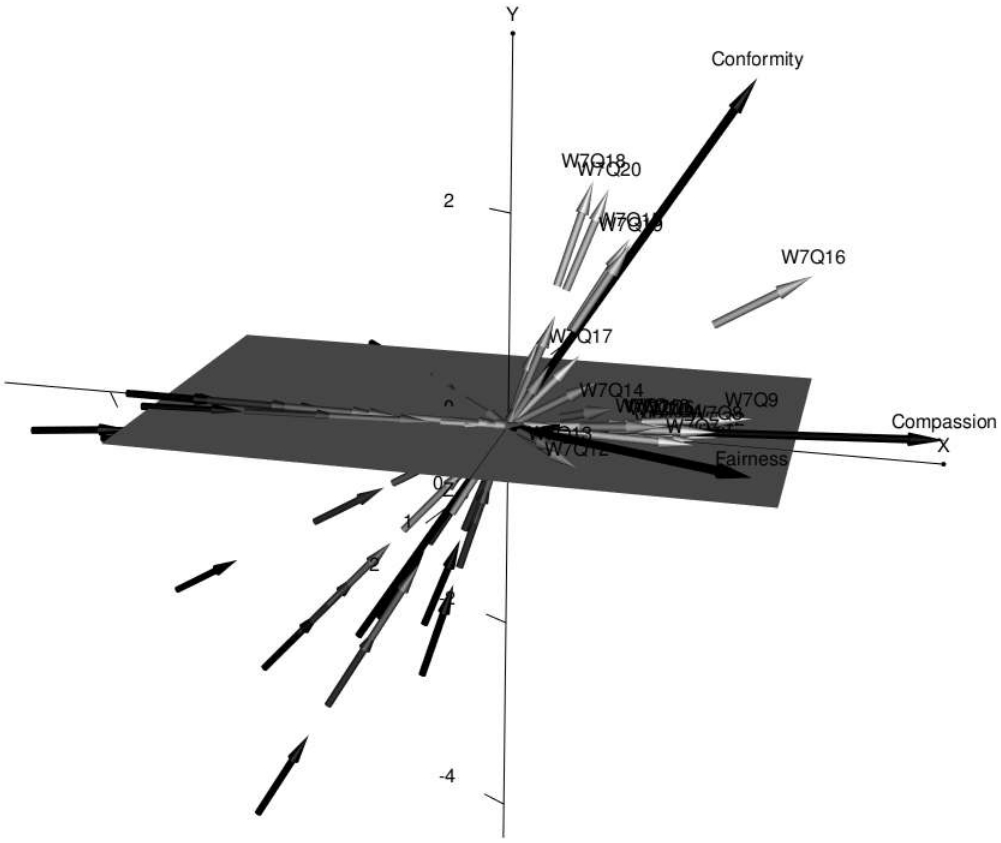

Figure 8: Model 3 rotated 15° displaying scaled items.

3.2. Profile Analysis

The `plot()` function can also display respondents in the three-dimensional model represented as spheres whose coordinates consist of the respondent's trait scores. This allows for plotting all respondent scores but also for performing a type of profile analysis in which respondents are selected and separated conditioned on single or multiple criteria. The resulting output shows where this subset of respondents is located in the model and what latent profile best describes them. Similarly, respondents sorted into, for instance, color categories can be simultaneously graphed to expose potential group-level effects.

3.2.1 Plotting All Respondents

To plot respondent scores using the default option, call `D3mirt()` and set `ind.scores` to `TRUE`. This will display respondents visualized as black spheres in the latent space. In the function call below, all items are hidden with `hide = TRUE` to reduce visual clutter, and `constructs` is set to FALSE (by default) for the same reason.

```
# Plot profiles with item vector arrows hidden
# Graphical output can be seen in Figure 9
plot(mod3, hide = TRUE, ind.scores = TRUE,
           x.lab = "Compassion", y.lab="Conformity", z.lab="Fairness")
```

Respondents can be separated by creating color categories for the spheres to aid the visual interpretation. This can be done using the `levels` argument in `plot()`. In brief, the `levels` argument calls `as.factor()` to count the number of factor levels in the data. This means that raw data can be used as is, but the number of colors in the color vectors argument (`sphere.col`) may need to be adapted. In the example below, the criteria variable for gender only holds two factor levels and, therefore, only two colors in the color vector are needed. To create an appropriate data frame for `levels`, the gender variable must first be assigned to a data frame.

```
# Load the data set as a matrix to remove attributes
data("anes0809offwaves")
x <- as.matrix(anes0809offwaves)
```

Call `plot()` using the respondent gender data located in column two in x, i.e., $x[, 2]$, in the `levels` argument. Please note that the criteria variable used in `levels` must follow the same row order as the frame used when fitting the `D3mirt`object. That is, rows must refer to the same respondents, otherwise the color matching will most likely be wrong.

```
# Plot profiles with item vector arrows hidden
# Score levels: 1 = Blue ("male") and 2 = Red ("female")
# Graphical output can be seen in Figure 10
plot(mod3, hide = TRUE, ind.scores = TRUE,
     levels = x[, 2], sphere.col = c("blue", "red"),
           x.lab = "Compassion", y.lab="Conformity", z.lab="Fairness")
```

An example of how the output can be described is as follows:

> In Figure 10, a simple gender profile can be observed, showing that more women tend to have higher levels of compassion. When rotating the model 90° clockwise, there seems to be no obvious gender difference related to Conformity or Fairness.

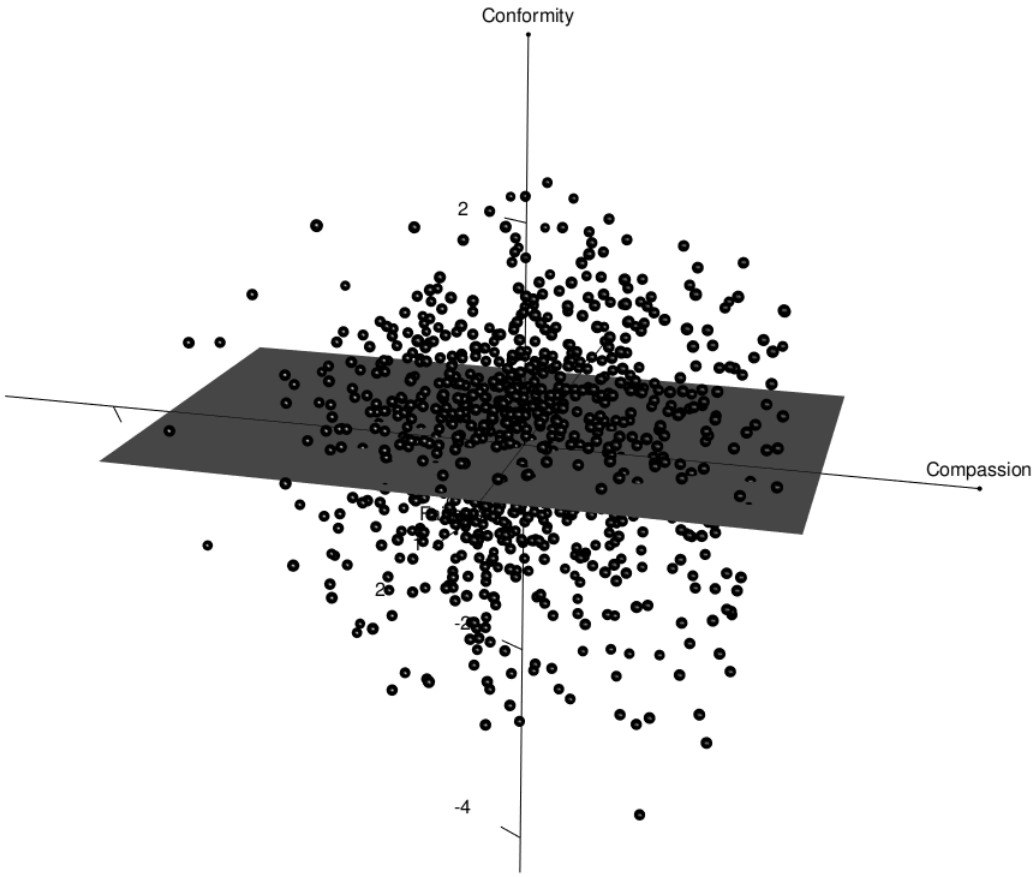

Figure 9: Model 3 rotated 15° showing all respondents trait scores illustrated as black spheres.

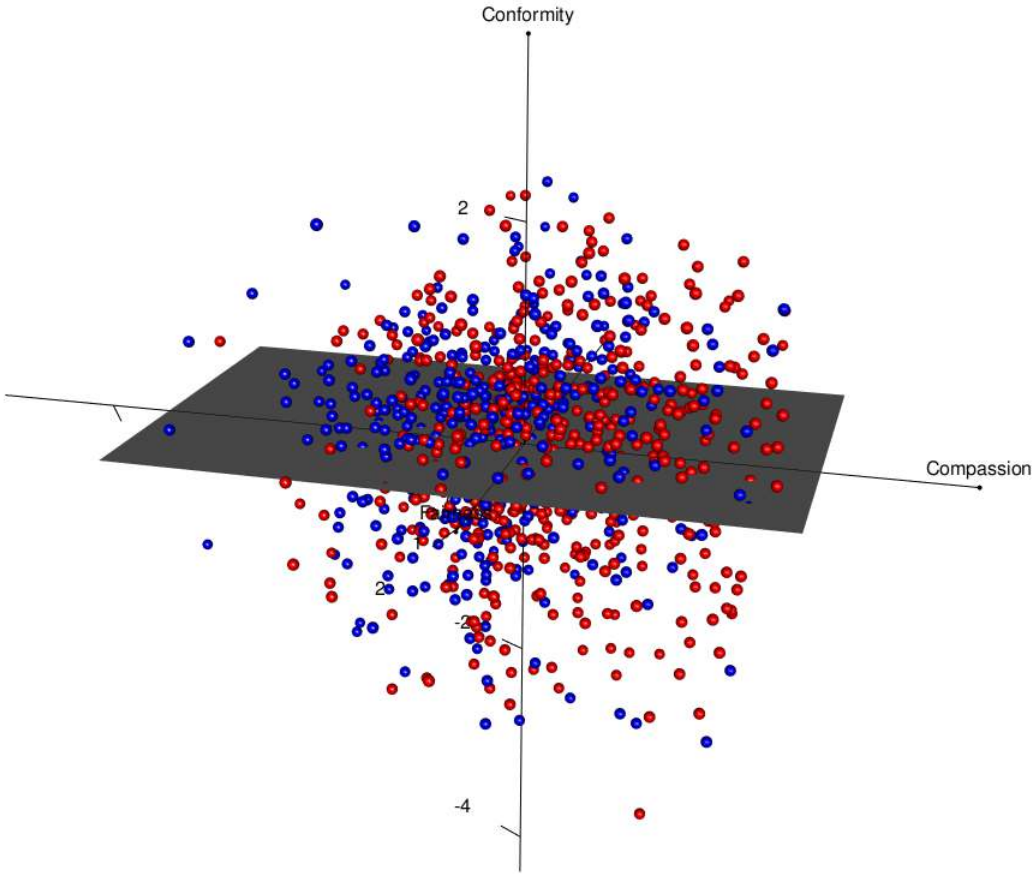

Figure 10: Model 3 rotated 15° graphically separating male trait scores (blue) from female trait scores (red).

3.2.2 Plotting the Orthogonal Model

If the `D3mirt` object was fitted using the orthogonal option, plotting all respondent scores can easily be done using the `ind.scores` argument set to `TRUE` as in the previous example. In the function call below, all arguments are the same, expcept that the `D3mirt` object is now the orthogonal version. As can be seen, plotting respondent trait scores using the orthogonal model slightly reduced the effect on gender seen when using the compensatory model. Note that this is not a general effect. Instead, the opposite can also be the case, i.e., that an effect appears because the model is not constrained to contain only strict unidimensions. It is, therefore, essential that the user justify using the orthogonal model because it is not, in itself, an empirical reflection of the data. It can, however, be a reflection of an idea.

```
# Plot profiles with item vector arrows hidden
# Score levels: 1 = Blue ("male") and 2 = Red ("female")
# Graphical output can be seen in Figure 11
plot(mod2, hide = TRUE, ind.scores = TRUE, levels = x[, 2], sphere.col = c("blue", "red"),
           x.lab = "Compassion", y.lab="Conformity", z.lab="Fairness"))
```

3.2.3 Plotting Subgroups of Respondents

It is also possible to plot a subset of respondents in the model space. To do this, the user must first create a data frame with respondent trait scores externally and use the same in the `profiles` argument in `plot()`. In the example below, this is done using the `rep()` function to create subgroups based on factor levels. This can be useful when a criterion variable, such as an age variable, has a wide data range. More specifically, a color vector for the `sphere.col` argument can be created with color names repeated by `rep()`. When plotting, the `plot()` function will then pick colors from the `sphere.col` argument following the factor order in the `levels` argument. Because the use of `rep()` repeats color labels, the first set of factor levels are colored with the same color, and the second set of factor levels are colored with some another color. The result will then show two groups in the latent space separated by two colors. The example below illustrates this approach for comparing respondents 30 years or younger against 70 years or older. Please note that when plotting subgroups, i.e., not the entire data frame of respondents, a data frame must be imputed into the `profiles` argument in `plot()`.

First, we must count the number of factor levels in the variable. This can be done with the `nlevels(as.factor())`. Then, we use the output from the latter to set the appropriate size of the color vector for `sphere.col`.

```
# Create a data frame containing trait scores from the D3mirt object and the age variable W3Xage
# This will result in three columns with trait scores and a fourth with age data
z <- data.frame(cbind(mod3$fscores, x[, 1]))

# Subset data frame z conditioned on age <= 30
z1 <- subset(z, z[, 4] <= 30)

# Subset data frame z conditioned on age >= 70
z2 <- subset(z, z[, 4] >= 70)

# Row bind z1 and z2
z <- rbind(z1,z2)

# Check number of factor levels with nlevels() and as.factor()
nlevels(as.factor(z1[, 4]))
#> [1] 14
```

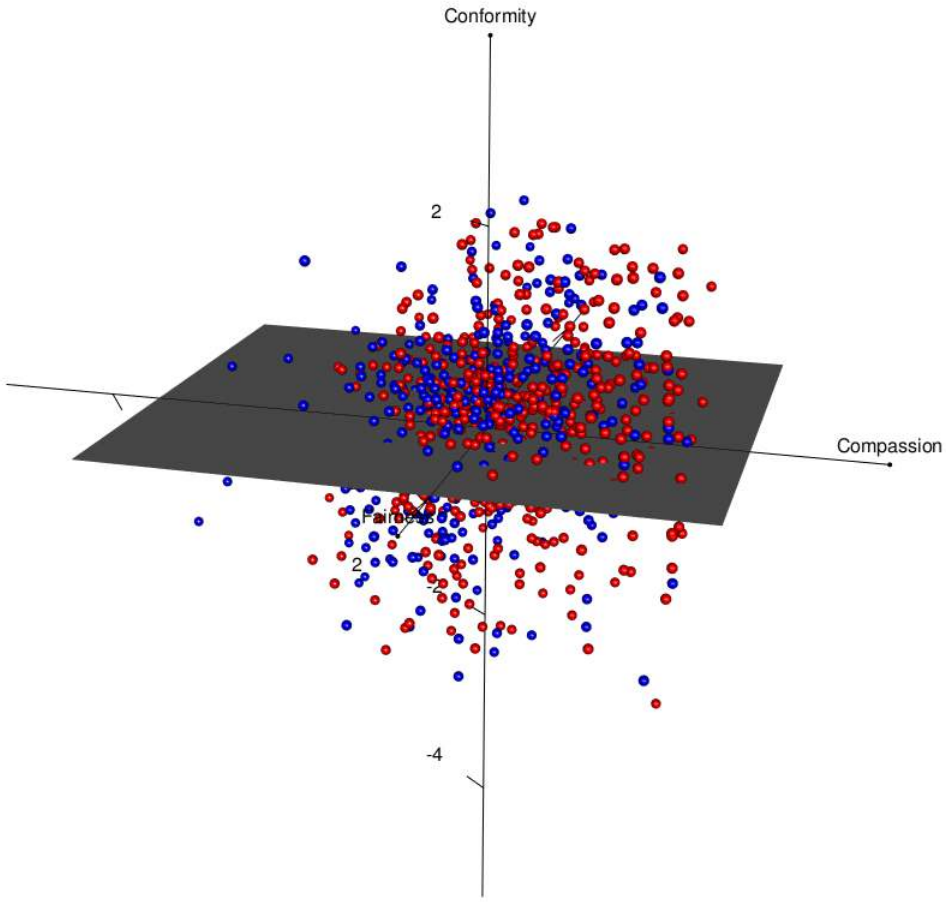

Figure 11: Model 2, the orthogonal model, rotated 15° showing respondents male trait scores (blue) and female trait scores (red).

```r
nlevels(as.factor(z2[, 4]))
#> [1] 16
```

```r
# Use rep() to create a color vector to color groups based on the nlevels() output
# z1 has 14 factor levels and z2 has 16 factor levels
# z1 respondents are colored red and z2 are colored blue
colvec <- c(rep("red", 14),
            rep("blue", 16))
```

```r
# Call plot() with profile data on age with item vector arrows hidden
# Use data frame z in the profiles argument and the criteria column
# from z in the levels argument
# Graphical output can be seen in Figure 12
plot(mod3, hide = TRUE, profiles = z, levels = z[, 4], sphere.col = colvec,
     x.lab = "Compassion", y.lab="Conformity", z.lab="Fairness")
```

An example of how the output can be described is as follows.

> As can be seen, Figure 12 indicates an age effect in which older individuals have higher levels of Conformity. Rotating the model to the left also shows that older individuals may have slightly higher levels of Fairness, even if the effect is less clear. There does not seem to be any difference in Compassion, however. To sum up, we can say, therefore, that the analysis indicates a possibility that older individuals may have a moral profile consisting of high Conformity and possibly also high Fairness.

3.2.3 Plotting Confidence Intervals

It is also possible to plot a confidence interval in the shape of an ellipse surrounding the spheres. The younger individuals (≤ 30) are selected and plotted with a 95% *CI* In the example below.

```r
# Column bind trait scores with the age variable W3Xage
z <- data.frame(cbind(mod3$fscores, x[, 1]))

# Subset data frame z conditioned on age <= 30
z1 <- subset(z, z[, 4] <= 30)

# Use rep() to create a color vector to color groups based on the nlevels() output
# z1 has 14 factor levels
colvec <- c(rep("red", 14))
```

To plot the *CI*, the `ci` argument is set to `TRUE`. The color of the spheres was also changed from the default `grey80` to `orange` in the example below. Note that the *CI* limit can be adjusted with the `ci.level` argument.

```r
# Call plot() with profile data on age with item vector arrows hidden
# Graphical output can be seen in Figure 13
plot(mod3, hide = TRUE, profiles = z1, levels = z1[, 4], sphere.col = colvec,
     x.lab = "Compassion", y.lab="Conformity", z.lab="Fairness",
     ci = TRUE, ci.level = 0.95, ellipse.col = "orange")
```

An example of how the output can be described is as follows.

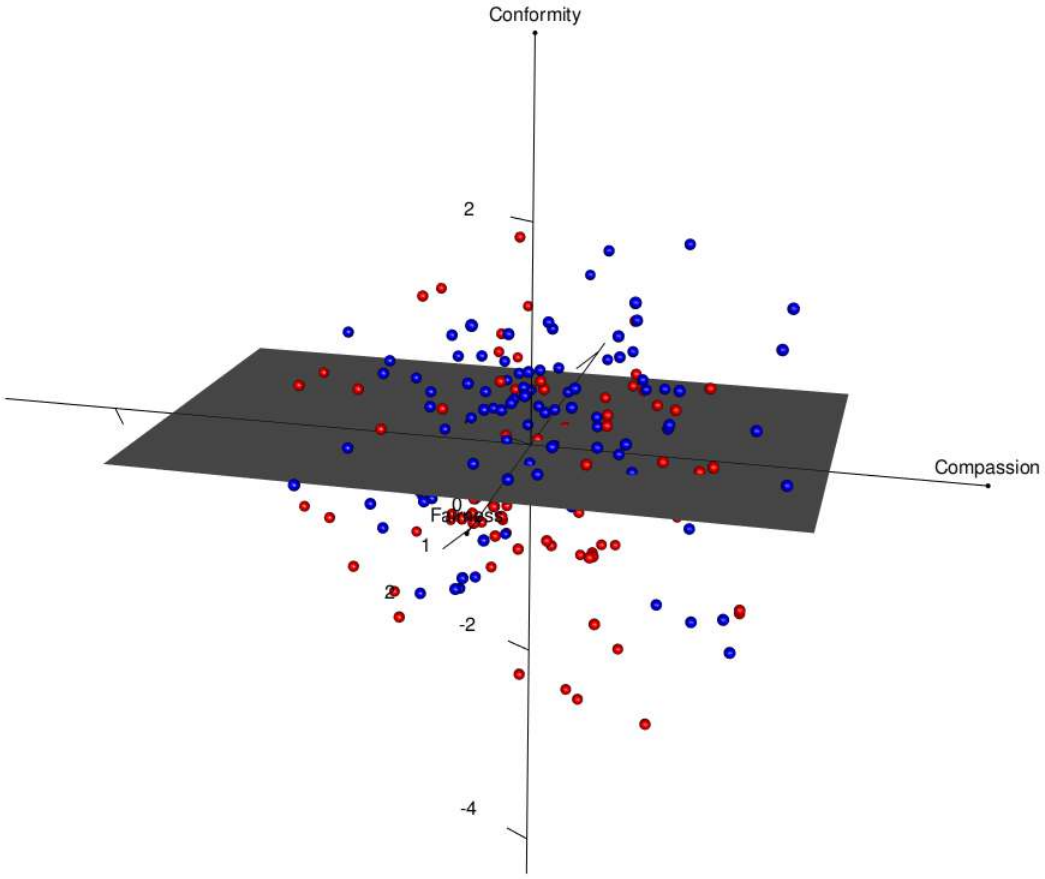

Figure 12: Model 3 rotated 15° graphically separating respondents 30 years or younger (red) from respondents 70 years and older (blue).

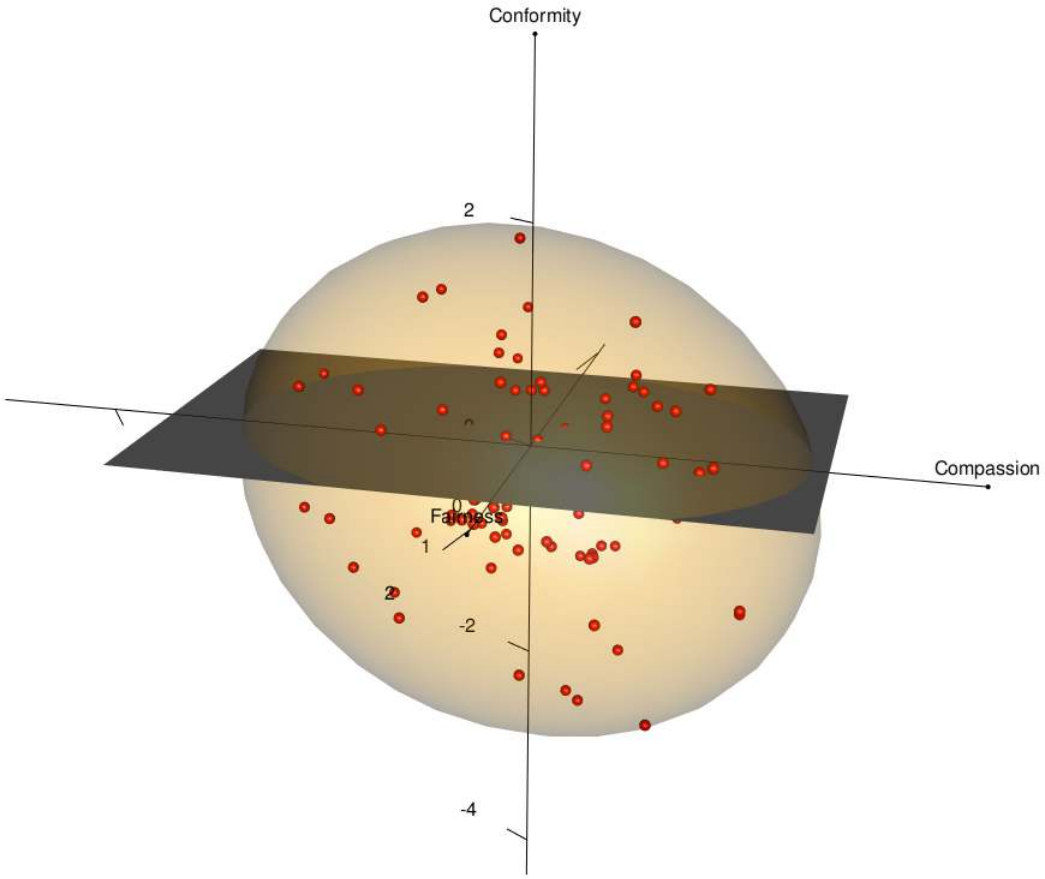

Figure 13: Model 3 rotated 15° showing respondents 30 years or younger surrounded by a 95 CI.

In Figure 13, we can see a tendency for a profile on age in which younger individuals could be described as less oriented towards Conformity. We can also observe a tendency for what could be an interaction effect in which higher levels of Conformity seem to be associated with lower levels of Fairness.

4. Exporting The RGL Device

Below are some options for exporting the RGL device. In addition, it is also possible to export graphical devices in R Markdown documents with `rgl::hook_webgl()` together with graphical options for knitr, as was done when creating this vignette.

```
# Export an open RGL device to the console that can be saved as an HTML or image file
plot(mod3, constructs = TRUE)
s <- rgl::scene3d()
rgl::rglwidget(s,
               width = 1040,
               height = 1040)

# Export a snap shoot of an open RGL device directly to file
plot(mod3, constructs = TRUE)
rgl::rgl.snapshot('RGLdevice.png',
                  fmt = 'png')
```

5. Getting Help, Feedback, and Questions

If you encounter a bug, please file an issue with a minimal reproducible example on GitHub (https://github.com/ForsbergPyschometrics/D3mirt). For questions and suggestions on how to improve the code, please contact me on GitHub or via email (forsbergpsychometrics@gmail.com).

6. References

Adler, D., & Murdoch, D. (2023). *Rgl: 3d Visualization Using OpenGL* [Computer software]. https://dmurdoch.github.io/rgl/index.html

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. https://doi.org/10.18637/jss.v048.i06

DeBell, M., Krosnick, J. A., & Lupia, A. (2010). *Methodology Report and User's Guide for the 2008-2009 ANES Panel Study.* Palo Alto, CA, and Ann Arbor, MI: Stanford University and the University of Michigan.

McKinley, R. L., & Reckase, M. D. (1983). *An Extension of the Two-parameter Logistic Model to the multidimensional latent space*, Report ONR83-2. Iowa City, IA, American College Testing Program.

Muraki, E., & Carlson, J. E. (1995). Full-Information Factor Analysis for Polytomous Item Responses. *Applied Psychological Measurement, 19*(1), 73-90. https://doi.org/10.1177/014662169501900109

Reckase, M. D. (2009). *Multidimensional Item Response Theory.*Springer. https://doi.org/10.1007/978-0-387-89976-3

Reckase, M. D. (1985). The Difficulty of Test Items That Measure More Than One Ability. *Applied Psychological Measurement, 9*(4),401-412. https://doi.org/10.1177/014662168500900409

Reckase, M. D., & McKinley, R. L. (1991). The Discriminating Power of Items That Measure More Than One Dimension. *Applied Psychological Measurement, 15*(4), 361-373. https://doi.org/10.1177/014662169101500407

7. Appendix A: Compensatory Model Syntax

The model syntax for the compensatory model and mirt code used in `D3mirt` can be seen below. This is the same syntax built into `D3mirt`, version $\geq$ 2.0.0. In brief, the three-dimensional compensatory model is specified so that all items load on all three factors in the model, and all factors are constrained to be orthogonal. Fitting the model is preferably done with the `mirt()` function from the package `mirt` (Chalmers, 2012).

Observe that the START and FIXED commands fix the slope parameters on the second, a_2, and third, a_3. factor for item$_1$ (W7Q3), and the slope on the third, a_3, factor for item i_2 (W7Q20). The resulting S4 object from `mirt` can be used in `D3mirt` in place of data frame x just in previous versions of the package.

```
# Load data
data("anes0809offwaves")
x <- anes0809offwaves
x <- x[, 3:22] # Remove columns for age and gender

# Fit a three-dimensional graded response model with orthogonal factors
# Example below uses Likert items from the built-in data set "anes0809offwaves"
# Item W7Q3 and item W7Q20 was selected with `modid()`
# The model specification allows all items in the data set (1-20)
# to load on all three factors (F1-F3)
# The START and FIXED commands are used on the two items to identify the DMIRT model
spec <-  ' F1 = 1-20
           F2 = 1-20
           F3 = 1-20

           START=(W7Q3,a2,0)
           START=(W7Q3,a3,0)

           START=(W7Q20,a3,0)

           FIXED=(W7Q3,a2)
           FIXED=(W7Q3,a3)

           FIXED=(W7Q20,a3) '

mod1 <- mirt::mirt(x,
                   spec,
                   itemtype = 'graded',
                   SE = TRUE,
                   method = 'QMCEM')
```