Tutorial for Chapter 4: BLOT Rasch Analysis with the eRm package

Tara L. Valladares & Moritz Heene

July 22, 2020

# Preliminary steps before you can run the Rasch model analysis

1. Create a folder called “BLOT\_Rasch” on your desktop, for example. Download the data set BLOT.csv and the R code Chapter\_4\_eRm.R from the website and save both in *that* folder. This folder will serve as your working directory containing all files you need to conduct the analysis and to store optional output (i.e., code, data, and figures). If the R code and the .csv files are not in the same folder, you will not be able to load the data using the code below.
2. Open the file Chapter\_4\_eRm.R in RStudio by clicking on the file. This will open the file in RStudio.
3. Go to “Session” “Set Working Directory” “To Source File Location”. This defines the folder you named above as your working directory in which you are currently working and where R expects all data sets to be.

You are now ready to run the Rasch model analysis of the BLOT data used in ARM4 Chapter 4. Please use the following instructions and explanations of the R code.

# Rasch Model Analysis: Instructions and Basic Explanations of the R Code

First, we will load the necessary packages we will need for this analysis. If you have not yet, you will need to install the packages using the install.packages() function below. This will only need to be done once. So, the first time you run the code just uncomment the line install.packages(c("eRm", "dplyr", "ggplot2", "pairwise", "TAM")) by deleting the ”#” symbol and rerun that line. This will install the packages. Then, we will load the packages into R using library(). You will need to run library() with each package you want to use each time you open R. If you encounter any errors, you may need to update R to the latest version.

# Installing the required packages for this analysis and the other tutorials.   
# This is only required once.   
# install.packages(c("eRm", "dplyr", "ggplot2", "pairwise", "TAM", "psych"))  
  
# Load the packages required for the analysis  
library(eRm)  
library(dplyr)

Notice that everything after a “#” symbol will be interpreted as a comment. R will not try to run commented code. You can add or remove the “#” symbol to tell R to run or not to run a certain line of code.

The code below says to read a .csv file into R and store it (<-) as a data frame (“df” for short). The file is called BLOT.csv, it has no header (header = FALSE) and a semi-colon is used as the field separator (sep = ";").

# Reading in the comma-seperated data set  
df <- read.csv("BLOT.csv", header = FALSE, sep = ";")

Before we can run the Rasch analysis we need to preprocess our data. The following lines of code define the column names of the data frame and finally create an object called “blot\_items” containing the BLOT items we need for the Rasch analysis. Each BLOT item is represented by a column. Each participant has their own row. There are 35 multiple-choice items administered to 150 persons. Each answer was scored 1 if correct, 0 if not correct. The BLOT test relates to Piagetian theory about cognitive development during adolescence.

First, we will use the dim function to look at the dimensions of our data frame - how many rows and columns it has. Then, we will add column names to the data frame. We have a couple extra columns in this data set, so we will just choose the 35 questions that are specific to the BLOT test. Finally, we will use the head function to see the first five rows of the blot\_items object.

# Look at the dimensions of "df" - we have 150 rows and 49 columns  
dim(df)

## [1] 150 49

# Defining the name of the first column of the data frame called "df"  
colnames(df)[1] <- c("id")   
# Defining the names of the columns referring to the items  
colnames(df)[2:49] <- paste("Blot", (1:48), sep="\_")   
# Selecting only the BLOT items  
blot\_items <- dplyr::select(df, Blot\_1:Blot\_35)   
# Looking at the first five rows of the dataframe  
head(blot\_items)

## Blot\_1 Blot\_2 Blot\_3 Blot\_4 Blot\_5 Blot\_6 Blot\_7 Blot\_8 Blot\_9 Blot\_10  
## 1 1 1 1 1 1 1 1 1 1 1  
## 2 1 1 1 1 1 1 1 1 1 1  
## 3 1 1 0 1 0 1 1 1 1 1  
## 4 1 1 1 1 1 1 1 1 1 1  
## 5 1 1 1 1 1 1 1 1 1 1  
## 6 1 1 1 1 1 1 1 1 1 1  
## Blot\_11 Blot\_12 Blot\_13 Blot\_14 Blot\_15 Blot\_16 Blot\_17 Blot\_18 Blot\_19  
## 1 0 1 1 0 1 0 1 1 0  
## 2 1 1 1 1 1 1 1 1 1  
## 3 1 1 1 1 0 1 1 1 1  
## 4 1 1 1 1 1 1 1 1 1  
## 5 1 1 0 1 1 1 1 1 1  
## 6 1 1 1 0 1 1 1 1 0  
## Blot\_20 Blot\_21 Blot\_22 Blot\_23 Blot\_24 Blot\_25 Blot\_26 Blot\_27 Blot\_28  
## 1 1 0 1 1 1 1 1 1 1  
## 2 1 1 1 1 1 1 1 1 0  
## 3 1 0 1 1 1 1 1 1 0  
## 4 1 1 0 1 1 1 1 1 1  
## 5 1 0 1 1 1 1 1 1 1  
## 6 1 0 1 1 1 1 1 1 1  
## Blot\_29 Blot\_30 Blot\_31 Blot\_32 Blot\_33 Blot\_34 Blot\_35  
## 1 1 0 1 1 1 1 1  
## 2 1 1 1 1 1 1 1  
## 3 1 0 1 1 1 1 1  
## 4 1 1 1 1 1 1 1  
## 5 1 1 1 1 1 1 1  
## 6 1 1 1 1 1 1 1

We are now ready to run the actual Rasch analysis. In accordance with the corresponding Winsteps analysis, we will

* Estimate model parameters
* Estimate item fit and person separation reliability
* Create a variable map
* Create a pathway map.

# Item Fit, Item Parameters, and Person Separation Reliability

We start by estimating the Rasch model parameters and storing them in an object called fit\_rasch\_blot. The following line uses the *eRm* package to run a basic Rasch model analysis of the data on the object blot\_items and save the results in an object called fit\_rasch\_blot.

# Estimates Rasch model parameters  
fit\_rasch\_blot <- eRm::RM(blot\_items)

If you ever want to know more about a function, you can look at the official documentation by using ? or searching in the Help tab in the files, plots, and packages pane. R documentation can seem intimidating at first, but you’ll quickly learn to rely on it! The entire package documentation can be found online at <https://cran.r-project.org/web/packages/eRm/eRm.pdf>

# Look at the documentation for the RM function  
?RM

Before we start interpreting model parameters (i.e., item difficulties, person parameters, and the Wright map), data-model fit needs to be assessed as there is no point in interpreting results if the data do not fit to the Rasch model.

In the following code, the person parameter for each person contained in our data set is estimated. These person parameters are needed as input for the item fit analysis.

# Estimating person parameters for each observed raw score.   
pparameters\_blot <- eRm::person.parameter(fit\_rasch\_blot)

In a second step, we reuse the previously estimated person parameters to estimate item fit statistics. We will look at it using the print function.

# Estimate item parameters  
item\_fit <- eRm::itemfit(pparameters\_blot)   
print(item\_fit)

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_1 98.894 146 0.999 0.673 0.966 -0.858 -0.156 0.413  
## Blot\_2 107.906 146 0.992 0.734 0.993 -0.686 0.012 0.395  
## Blot\_3 132.504 146 0.781 0.901 0.978 -0.563 -0.235 0.465  
## Blot\_4 127.534 146 0.862 0.868 0.992 -0.479 -0.034 0.426  
## Blot\_5 108.213 146 0.992 0.736 0.956 -0.547 -0.184 0.377  
## Blot\_6 114.442 146 0.975 0.779 1.007 0.031 0.145 0.241  
## Blot\_7 93.946 146 1.000 0.639 0.958 -1.065 -0.228 0.420  
## Blot\_8 147.396 146 0.452 1.003 0.912 0.069 -1.092 0.547  
## Blot\_9 141.821 146 0.582 0.965 1.061 -0.085 0.614 0.379  
## Blot\_10 99.791 146 0.999 0.679 0.907 -1.228 -0.742 0.502  
## Blot\_11 140.362 146 0.616 0.955 1.017 -0.131 0.200 0.412  
## Blot\_12 34.257 146 1.000 0.233 0.664 -1.598 -1.316 0.623  
## Blot\_13 193.280 146 0.005 1.315 1.162 2.002 1.971 0.311  
## Blot\_14 188.287 146 0.011 1.281 1.134 0.830 0.863 0.221  
## Blot\_15 123.802 146 0.909 0.842 0.965 -1.089 -0.421 0.481  
## Blot\_16 148.869 146 0.418 1.013 1.117 0.141 0.918 0.296  
## Blot\_17 109.310 146 0.990 0.744 0.865 -1.352 -1.475 0.567  
## Blot\_18 107.506 146 0.993 0.731 0.893 -1.087 -0.935 0.518  
## Blot\_19 153.222 146 0.325 1.042 1.009 0.283 0.126 0.430  
## Blot\_20 116.335 146 0.966 0.791 0.894 -0.444 -0.592 0.466  
## Blot\_21 243.879 146 0.000 1.659 1.248 3.523 2.529 0.169  
## Blot\_22 236.313 146 0.000 1.608 0.888 1.313 -0.545 0.433  
## Blot\_23 135.256 146 0.728 0.920 1.057 -0.336 0.615 0.383  
## Blot\_24 149.846 146 0.397 1.019 0.887 0.161 -1.125 0.535  
## Blot\_25 183.566 146 0.019 1.249 1.070 1.300 0.791 0.359  
## Blot\_26 110.900 146 0.986 0.754 0.896 -1.605 -1.275 0.530  
## Blot\_27 88.838 146 1.000 0.604 0.829 -1.004 -0.975 0.515  
## Blot\_28 179.678 146 0.030 1.222 1.110 1.649 1.360 0.346  
## Blot\_29 103.054 146 0.997 0.701 0.924 -0.944 -0.512 0.467  
## Blot\_30 168.153 146 0.101 1.144 1.185 1.011 2.248 0.283  
## Blot\_31 224.002 146 0.000 1.524 1.065 2.097 0.650 0.359  
## Blot\_32 124.722 146 0.898 0.848 0.958 -1.103 -0.523 0.500  
## Blot\_33 134.675 146 0.739 0.916 1.079 -0.152 0.583 0.308  
## Blot\_34 113.985 146 0.977 0.775 0.988 -0.685 -0.044 0.422  
## Blot\_35 106.324 146 0.994 0.723 0.915 -0.957 -0.630 0.496

Notice that the *eRm* item fit output somewhat deviates from Winsteps’ item fit report. While they both report the unstandardized and standardized MNSQ infit and outfit statistics, *eRm* also provides chi-square based itemfit statistics. The column Discrim shows the corrected item-raw score correlations correcting for item overlap as well as for the drop in score reliability when an item is removed from the scale to calculate item-total correlations (see Cureton, 1966, for details).

*eRm* also provides additional function arguments to sort the output according to any of the columns p-value, Outfit MSQ, Infit MSQ, Outfit t, Infit t, or Discrim. So, let us say we want to sort the table according to the infit *t*-statistic because this would make it easier to spot possibly misfitting items. To do so, we add sort\_by = "infit\_t" to the print(item\_fit) function as illustrated in the following example:

print(item\_fit, sort\_by = "infit\_t")

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_2 107.906 146 0.992 0.734 0.993 -0.686 0.012 0.395  
## Blot\_4 127.534 146 0.862 0.868 0.992 -0.479 -0.034 0.426  
## Blot\_34 113.985 146 0.977 0.775 0.988 -0.685 -0.044 0.422  
## Blot\_19 153.222 146 0.325 1.042 1.009 0.283 0.126 0.430  
## Blot\_6 114.442 146 0.975 0.779 1.007 0.031 0.145 0.241  
## Blot\_1 98.894 146 0.999 0.673 0.966 -0.858 -0.156 0.413  
## Blot\_5 108.213 146 0.992 0.736 0.956 -0.547 -0.184 0.377  
## Blot\_11 140.362 146 0.616 0.955 1.017 -0.131 0.200 0.412  
## Blot\_7 93.946 146 1.000 0.639 0.958 -1.065 -0.228 0.420  
## Blot\_3 132.504 146 0.781 0.901 0.978 -0.563 -0.235 0.465  
## Blot\_15 123.802 146 0.909 0.842 0.965 -1.089 -0.421 0.481  
## Blot\_29 103.054 146 0.997 0.701 0.924 -0.944 -0.512 0.467  
## Blot\_32 124.722 146 0.898 0.848 0.958 -1.103 -0.523 0.500  
## Blot\_22 236.313 146 0.000 1.608 0.888 1.313 -0.545 0.433  
## Blot\_33 134.675 146 0.739 0.916 1.079 -0.152 0.583 0.308  
## Blot\_20 116.335 146 0.966 0.791 0.894 -0.444 -0.592 0.466  
## Blot\_9 141.821 146 0.582 0.965 1.061 -0.085 0.614 0.379  
## Blot\_23 135.256 146 0.728 0.920 1.057 -0.336 0.615 0.383  
## Blot\_35 106.324 146 0.994 0.723 0.915 -0.957 -0.630 0.496  
## Blot\_31 224.002 146 0.000 1.524 1.065 2.097 0.650 0.359  
## Blot\_10 99.791 146 0.999 0.679 0.907 -1.228 -0.742 0.502  
## Blot\_25 183.566 146 0.019 1.249 1.070 1.300 0.791 0.359  
## Blot\_14 188.287 146 0.011 1.281 1.134 0.830 0.863 0.221  
## Blot\_16 148.869 146 0.418 1.013 1.117 0.141 0.918 0.296  
## Blot\_18 107.506 146 0.993 0.731 0.893 -1.087 -0.935 0.518  
## Blot\_27 88.838 146 1.000 0.604 0.829 -1.004 -0.975 0.515  
## Blot\_8 147.396 146 0.452 1.003 0.912 0.069 -1.092 0.547  
## Blot\_24 149.846 146 0.397 1.019 0.887 0.161 -1.125 0.535  
## Blot\_26 110.900 146 0.986 0.754 0.896 -1.605 -1.275 0.530  
## Blot\_12 34.257 146 1.000 0.233 0.664 -1.598 -1.316 0.623  
## Blot\_28 179.678 146 0.030 1.222 1.110 1.649 1.360 0.346  
## Blot\_17 109.310 146 0.990 0.744 0.865 -1.352 -1.475 0.567  
## Blot\_13 193.280 146 0.005 1.315 1.162 2.002 1.971 0.311  
## Blot\_30 168.153 146 0.101 1.144 1.185 1.011 2.248 0.283  
## Blot\_21 243.879 146 0.000 1.659 1.248 3.523 2.529 0.169

The default setting sorts the items in an *increasing* order according to the specified criterion. For t-statistics, the items are sorted in increasing order of magnitude, *ignoring* the sign. If you want to sort the output table, for example, in a *decreasing* order with possibly misfitting items at the top of the table, just add the argument decreasing = TRUE to the previous command as follows:

print(item\_fit, sort\_by = "infit\_t", decreasing = TRUE)

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_21 243.879 146 0.000 1.659 1.248 3.523 2.529 0.169  
## Blot\_30 168.153 146 0.101 1.144 1.185 1.011 2.248 0.283  
## Blot\_13 193.280 146 0.005 1.315 1.162 2.002 1.971 0.311  
## Blot\_17 109.310 146 0.990 0.744 0.865 -1.352 -1.475 0.567  
## Blot\_28 179.678 146 0.030 1.222 1.110 1.649 1.360 0.346  
## Blot\_12 34.257 146 1.000 0.233 0.664 -1.598 -1.316 0.623  
## Blot\_26 110.900 146 0.986 0.754 0.896 -1.605 -1.275 0.530  
## Blot\_24 149.846 146 0.397 1.019 0.887 0.161 -1.125 0.535  
## Blot\_8 147.396 146 0.452 1.003 0.912 0.069 -1.092 0.547  
## Blot\_27 88.838 146 1.000 0.604 0.829 -1.004 -0.975 0.515  
## Blot\_18 107.506 146 0.993 0.731 0.893 -1.087 -0.935 0.518  
## Blot\_16 148.869 146 0.418 1.013 1.117 0.141 0.918 0.296  
## Blot\_14 188.287 146 0.011 1.281 1.134 0.830 0.863 0.221  
## Blot\_25 183.566 146 0.019 1.249 1.070 1.300 0.791 0.359  
## Blot\_10 99.791 146 0.999 0.679 0.907 -1.228 -0.742 0.502  
## Blot\_31 224.002 146 0.000 1.524 1.065 2.097 0.650 0.359  
## Blot\_35 106.324 146 0.994 0.723 0.915 -0.957 -0.630 0.496  
## Blot\_23 135.256 146 0.728 0.920 1.057 -0.336 0.615 0.383  
## Blot\_9 141.821 146 0.582 0.965 1.061 -0.085 0.614 0.379  
## Blot\_20 116.335 146 0.966 0.791 0.894 -0.444 -0.592 0.466  
## Blot\_33 134.675 146 0.739 0.916 1.079 -0.152 0.583 0.308  
## Blot\_22 236.313 146 0.000 1.608 0.888 1.313 -0.545 0.433  
## Blot\_32 124.722 146 0.898 0.848 0.958 -1.103 -0.523 0.500  
## Blot\_29 103.054 146 0.997 0.701 0.924 -0.944 -0.512 0.467  
## Blot\_15 123.802 146 0.909 0.842 0.965 -1.089 -0.421 0.481  
## Blot\_3 132.504 146 0.781 0.901 0.978 -0.563 -0.235 0.465  
## Blot\_7 93.946 146 1.000 0.639 0.958 -1.065 -0.228 0.420  
## Blot\_11 140.362 146 0.616 0.955 1.017 -0.131 0.200 0.412  
## Blot\_5 108.213 146 0.992 0.736 0.956 -0.547 -0.184 0.377  
## Blot\_1 98.894 146 0.999 0.673 0.966 -0.858 -0.156 0.413  
## Blot\_6 114.442 146 0.975 0.779 1.007 0.031 0.145 0.241  
## Blot\_19 153.222 146 0.325 1.042 1.009 0.283 0.126 0.430  
## Blot\_34 113.985 146 0.977 0.775 0.988 -0.685 -0.044 0.422  
## Blot\_4 127.534 146 0.862 0.868 0.992 -0.479 -0.034 0.426  
## Blot\_2 107.906 146 0.992 0.734 0.993 -0.686 0.012 0.395

Now, let us play around with the options to explore the functionality of this function:

1. Sort the table by decreasing MNSQ infit values:

print(item\_fit, sort\_by = "infit\_MSQ", decreasing = TRUE)

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_21 243.879 146 0.000 1.659 1.248 3.523 2.529 0.169  
## Blot\_30 168.153 146 0.101 1.144 1.185 1.011 2.248 0.283  
## Blot\_13 193.280 146 0.005 1.315 1.162 2.002 1.971 0.311  
## Blot\_14 188.287 146 0.011 1.281 1.134 0.830 0.863 0.221  
## Blot\_16 148.869 146 0.418 1.013 1.117 0.141 0.918 0.296  
## Blot\_28 179.678 146 0.030 1.222 1.110 1.649 1.360 0.346  
## Blot\_33 134.675 146 0.739 0.916 1.079 -0.152 0.583 0.308  
## Blot\_25 183.566 146 0.019 1.249 1.070 1.300 0.791 0.359  
## Blot\_31 224.002 146 0.000 1.524 1.065 2.097 0.650 0.359  
## Blot\_9 141.821 146 0.582 0.965 1.061 -0.085 0.614 0.379  
## Blot\_23 135.256 146 0.728 0.920 1.057 -0.336 0.615 0.383  
## Blot\_11 140.362 146 0.616 0.955 1.017 -0.131 0.200 0.412  
## Blot\_19 153.222 146 0.325 1.042 1.009 0.283 0.126 0.430  
## Blot\_6 114.442 146 0.975 0.779 1.007 0.031 0.145 0.241  
## Blot\_2 107.906 146 0.992 0.734 0.993 -0.686 0.012 0.395  
## Blot\_4 127.534 146 0.862 0.868 0.992 -0.479 -0.034 0.426  
## Blot\_34 113.985 146 0.977 0.775 0.988 -0.685 -0.044 0.422  
## Blot\_3 132.504 146 0.781 0.901 0.978 -0.563 -0.235 0.465  
## Blot\_1 98.894 146 0.999 0.673 0.966 -0.858 -0.156 0.413  
## Blot\_15 123.802 146 0.909 0.842 0.965 -1.089 -0.421 0.481  
## Blot\_7 93.946 146 1.000 0.639 0.958 -1.065 -0.228 0.420  
## Blot\_32 124.722 146 0.898 0.848 0.958 -1.103 -0.523 0.500  
## Blot\_5 108.213 146 0.992 0.736 0.956 -0.547 -0.184 0.377  
## Blot\_29 103.054 146 0.997 0.701 0.924 -0.944 -0.512 0.467  
## Blot\_35 106.324 146 0.994 0.723 0.915 -0.957 -0.630 0.496  
## Blot\_8 147.396 146 0.452 1.003 0.912 0.069 -1.092 0.547  
## Blot\_10 99.791 146 0.999 0.679 0.907 -1.228 -0.742 0.502  
## Blot\_26 110.900 146 0.986 0.754 0.896 -1.605 -1.275 0.530  
## Blot\_20 116.335 146 0.966 0.791 0.894 -0.444 -0.592 0.466  
## Blot\_18 107.506 146 0.993 0.731 0.893 -1.087 -0.935 0.518  
## Blot\_22 236.313 146 0.000 1.608 0.888 1.313 -0.545 0.433  
## Blot\_24 149.846 146 0.397 1.019 0.887 0.161 -1.125 0.535  
## Blot\_17 109.310 146 0.990 0.744 0.865 -1.352 -1.475 0.567  
## Blot\_27 88.838 146 1.000 0.604 0.829 -1.004 -0.975 0.515  
## Blot\_12 34.257 146 1.000 0.233 0.664 -1.598 -1.316 0.623

1. Sort the table by increasing MNSQ outfit *t*-values:

print(item\_fit, sort\_by = "outfit\_t")

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_6 114.442 146 0.975 0.779 1.007 0.031 0.145 0.241  
## Blot\_8 147.396 146 0.452 1.003 0.912 0.069 -1.092 0.547  
## Blot\_9 141.821 146 0.582 0.965 1.061 -0.085 0.614 0.379  
## Blot\_11 140.362 146 0.616 0.955 1.017 -0.131 0.200 0.412  
## Blot\_16 148.869 146 0.418 1.013 1.117 0.141 0.918 0.296  
## Blot\_33 134.675 146 0.739 0.916 1.079 -0.152 0.583 0.308  
## Blot\_24 149.846 146 0.397 1.019 0.887 0.161 -1.125 0.535  
## Blot\_19 153.222 146 0.325 1.042 1.009 0.283 0.126 0.430  
## Blot\_23 135.256 146 0.728 0.920 1.057 -0.336 0.615 0.383  
## Blot\_20 116.335 146 0.966 0.791 0.894 -0.444 -0.592 0.466  
## Blot\_4 127.534 146 0.862 0.868 0.992 -0.479 -0.034 0.426  
## Blot\_5 108.213 146 0.992 0.736 0.956 -0.547 -0.184 0.377  
## Blot\_3 132.504 146 0.781 0.901 0.978 -0.563 -0.235 0.465  
## Blot\_34 113.985 146 0.977 0.775 0.988 -0.685 -0.044 0.422  
## Blot\_2 107.906 146 0.992 0.734 0.993 -0.686 0.012 0.395  
## Blot\_14 188.287 146 0.011 1.281 1.134 0.830 0.863 0.221  
## Blot\_1 98.894 146 0.999 0.673 0.966 -0.858 -0.156 0.413  
## Blot\_29 103.054 146 0.997 0.701 0.924 -0.944 -0.512 0.467  
## Blot\_35 106.324 146 0.994 0.723 0.915 -0.957 -0.630 0.496  
## Blot\_27 88.838 146 1.000 0.604 0.829 -1.004 -0.975 0.515  
## Blot\_30 168.153 146 0.101 1.144 1.185 1.011 2.248 0.283  
## Blot\_7 93.946 146 1.000 0.639 0.958 -1.065 -0.228 0.420  
## Blot\_18 107.506 146 0.993 0.731 0.893 -1.087 -0.935 0.518  
## Blot\_15 123.802 146 0.909 0.842 0.965 -1.089 -0.421 0.481  
## Blot\_32 124.722 146 0.898 0.848 0.958 -1.103 -0.523 0.500  
## Blot\_10 99.791 146 0.999 0.679 0.907 -1.228 -0.742 0.502  
## Blot\_25 183.566 146 0.019 1.249 1.070 1.300 0.791 0.359  
## Blot\_22 236.313 146 0.000 1.608 0.888 1.313 -0.545 0.433  
## Blot\_17 109.310 146 0.990 0.744 0.865 -1.352 -1.475 0.567  
## Blot\_12 34.257 146 1.000 0.233 0.664 -1.598 -1.316 0.623  
## Blot\_26 110.900 146 0.986 0.754 0.896 -1.605 -1.275 0.530  
## Blot\_28 179.678 146 0.030 1.222 1.110 1.649 1.360 0.346  
## Blot\_13 193.280 146 0.005 1.315 1.162 2.002 1.971 0.311  
## Blot\_31 224.002 146 0.000 1.524 1.065 2.097 0.650 0.359  
## Blot\_21 243.879 146 0.000 1.659 1.248 3.523 2.529 0.169

1. Sort the table by decreasing MNSQ outfit values:

print(item\_fit, sort\_by = "outfit\_MSQ", decreasing = TRUE)

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_21 243.879 146 0.000 1.659 1.248 3.523 2.529 0.169  
## Blot\_22 236.313 146 0.000 1.608 0.888 1.313 -0.545 0.433  
## Blot\_31 224.002 146 0.000 1.524 1.065 2.097 0.650 0.359  
## Blot\_13 193.280 146 0.005 1.315 1.162 2.002 1.971 0.311  
## Blot\_14 188.287 146 0.011 1.281 1.134 0.830 0.863 0.221  
## Blot\_25 183.566 146 0.019 1.249 1.070 1.300 0.791 0.359  
## Blot\_28 179.678 146 0.030 1.222 1.110 1.649 1.360 0.346  
## Blot\_30 168.153 146 0.101 1.144 1.185 1.011 2.248 0.283  
## Blot\_19 153.222 146 0.325 1.042 1.009 0.283 0.126 0.430  
## Blot\_24 149.846 146 0.397 1.019 0.887 0.161 -1.125 0.535  
## Blot\_16 148.869 146 0.418 1.013 1.117 0.141 0.918 0.296  
## Blot\_8 147.396 146 0.452 1.003 0.912 0.069 -1.092 0.547  
## Blot\_9 141.821 146 0.582 0.965 1.061 -0.085 0.614 0.379  
## Blot\_11 140.362 146 0.616 0.955 1.017 -0.131 0.200 0.412  
## Blot\_23 135.256 146 0.728 0.920 1.057 -0.336 0.615 0.383  
## Blot\_33 134.675 146 0.739 0.916 1.079 -0.152 0.583 0.308  
## Blot\_3 132.504 146 0.781 0.901 0.978 -0.563 -0.235 0.465  
## Blot\_4 127.534 146 0.862 0.868 0.992 -0.479 -0.034 0.426  
## Blot\_32 124.722 146 0.898 0.848 0.958 -1.103 -0.523 0.500  
## Blot\_15 123.802 146 0.909 0.842 0.965 -1.089 -0.421 0.481  
## Blot\_20 116.335 146 0.966 0.791 0.894 -0.444 -0.592 0.466  
## Blot\_6 114.442 146 0.975 0.779 1.007 0.031 0.145 0.241  
## Blot\_34 113.985 146 0.977 0.775 0.988 -0.685 -0.044 0.422  
## Blot\_26 110.900 146 0.986 0.754 0.896 -1.605 -1.275 0.530  
## Blot\_17 109.310 146 0.990 0.744 0.865 -1.352 -1.475 0.567  
## Blot\_5 108.213 146 0.992 0.736 0.956 -0.547 -0.184 0.377  
## Blot\_2 107.906 146 0.992 0.734 0.993 -0.686 0.012 0.395  
## Blot\_18 107.506 146 0.993 0.731 0.893 -1.087 -0.935 0.518  
## Blot\_35 106.324 146 0.994 0.723 0.915 -0.957 -0.630 0.496  
## Blot\_29 103.054 146 0.997 0.701 0.924 -0.944 -0.512 0.467  
## Blot\_10 99.791 146 0.999 0.679 0.907 -1.228 -0.742 0.502  
## Blot\_1 98.894 146 0.999 0.673 0.966 -0.858 -0.156 0.413  
## Blot\_7 93.946 146 1.000 0.639 0.958 -1.065 -0.228 0.420  
## Blot\_27 88.838 146 1.000 0.604 0.829 -1.004 -0.975 0.515  
## Blot\_12 34.257 146 1.000 0.233 0.664 -1.598 -1.316 0.623

1. Sort the table by increasing *p*-values of the chi-square model test with two digits:

print(item\_fit, sort\_by = "p", digits = 2)

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_21 243.88 146 0.00 1.66 1.25 3.52 2.53 0.17  
## Blot\_22 236.31 146 0.00 1.61 0.89 1.31 -0.55 0.43  
## Blot\_31 224.00 146 0.00 1.52 1.06 2.10 0.65 0.36  
## Blot\_13 193.28 146 0.01 1.31 1.16 2.00 1.97 0.31  
## Blot\_14 188.29 146 0.01 1.28 1.13 0.83 0.86 0.22  
## Blot\_25 183.57 146 0.02 1.25 1.07 1.30 0.79 0.36  
## Blot\_28 179.68 146 0.03 1.22 1.11 1.65 1.36 0.35  
## Blot\_30 168.15 146 0.10 1.14 1.19 1.01 2.25 0.28  
## Blot\_19 153.22 146 0.32 1.04 1.01 0.28 0.13 0.43  
## Blot\_24 149.85 146 0.40 1.02 0.89 0.16 -1.13 0.53  
## Blot\_16 148.87 146 0.42 1.01 1.12 0.14 0.92 0.30  
## Blot\_8 147.40 146 0.45 1.00 0.91 0.07 -1.09 0.55  
## Blot\_9 141.82 146 0.58 0.96 1.06 -0.08 0.61 0.38  
## Blot\_11 140.36 146 0.62 0.95 1.02 -0.13 0.20 0.41  
## Blot\_23 135.26 146 0.73 0.92 1.06 -0.34 0.61 0.38  
## Blot\_33 134.68 146 0.74 0.92 1.08 -0.15 0.58 0.31  
## Blot\_3 132.50 146 0.78 0.90 0.98 -0.56 -0.24 0.46  
## Blot\_4 127.53 146 0.86 0.87 0.99 -0.48 -0.03 0.43  
## Blot\_32 124.72 146 0.90 0.85 0.96 -1.10 -0.52 0.50  
## Blot\_15 123.80 146 0.91 0.84 0.96 -1.09 -0.42 0.48  
## Blot\_6 114.44 146 0.97 0.78 1.01 0.03 0.15 0.24  
## Blot\_20 116.34 146 0.97 0.79 0.89 -0.44 -0.59 0.47  
## Blot\_34 113.99 146 0.98 0.78 0.99 -0.69 -0.04 0.42  
## Blot\_2 107.91 146 0.99 0.73 0.99 -0.69 0.01 0.39  
## Blot\_5 108.21 146 0.99 0.74 0.96 -0.55 -0.18 0.38  
## Blot\_17 109.31 146 0.99 0.74 0.86 -1.35 -1.48 0.57  
## Blot\_18 107.51 146 0.99 0.73 0.89 -1.09 -0.94 0.52  
## Blot\_26 110.90 146 0.99 0.75 0.90 -1.61 -1.28 0.53  
## Blot\_35 106.32 146 0.99 0.72 0.92 -0.96 -0.63 0.50  
## Blot\_1 98.89 146 1.00 0.67 0.97 -0.86 -0.16 0.41  
## Blot\_7 93.95 146 1.00 0.64 0.96 -1.07 -0.23 0.42  
## Blot\_10 99.79 146 1.00 0.68 0.91 -1.23 -0.74 0.50  
## Blot\_12 34.26 146 1.00 0.23 0.66 -1.60 -1.32 0.62  
## Blot\_27 88.84 146 1.00 0.60 0.83 -1.00 -0.97 0.52  
## Blot\_29 103.05 146 1.00 0.70 0.92 -0.94 -0.51 0.47

1. Sort the table by *decreasing* *p*-values of the chi-square model test:

print(item\_fit, sort\_by = "p", decreasing = TRUE)

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_7 93.946 146 1.000 0.639 0.958 -1.065 -0.228 0.420  
## Blot\_12 34.257 146 1.000 0.233 0.664 -1.598 -1.316 0.623  
## Blot\_27 88.838 146 1.000 0.604 0.829 -1.004 -0.975 0.515  
## Blot\_1 98.894 146 0.999 0.673 0.966 -0.858 -0.156 0.413  
## Blot\_10 99.791 146 0.999 0.679 0.907 -1.228 -0.742 0.502  
## Blot\_29 103.054 146 0.997 0.701 0.924 -0.944 -0.512 0.467  
## Blot\_35 106.324 146 0.994 0.723 0.915 -0.957 -0.630 0.496  
## Blot\_18 107.506 146 0.993 0.731 0.893 -1.087 -0.935 0.518  
## Blot\_2 107.906 146 0.992 0.734 0.993 -0.686 0.012 0.395  
## Blot\_5 108.213 146 0.992 0.736 0.956 -0.547 -0.184 0.377  
## Blot\_17 109.310 146 0.990 0.744 0.865 -1.352 -1.475 0.567  
## Blot\_26 110.900 146 0.986 0.754 0.896 -1.605 -1.275 0.530  
## Blot\_34 113.985 146 0.977 0.775 0.988 -0.685 -0.044 0.422  
## Blot\_6 114.442 146 0.975 0.779 1.007 0.031 0.145 0.241  
## Blot\_20 116.335 146 0.966 0.791 0.894 -0.444 -0.592 0.466  
## Blot\_15 123.802 146 0.909 0.842 0.965 -1.089 -0.421 0.481  
## Blot\_32 124.722 146 0.898 0.848 0.958 -1.103 -0.523 0.500  
## Blot\_4 127.534 146 0.862 0.868 0.992 -0.479 -0.034 0.426  
## Blot\_3 132.504 146 0.781 0.901 0.978 -0.563 -0.235 0.465  
## Blot\_33 134.675 146 0.739 0.916 1.079 -0.152 0.583 0.308  
## Blot\_23 135.256 146 0.728 0.920 1.057 -0.336 0.615 0.383  
## Blot\_11 140.362 146 0.616 0.955 1.017 -0.131 0.200 0.412  
## Blot\_9 141.821 146 0.582 0.965 1.061 -0.085 0.614 0.379  
## Blot\_8 147.396 146 0.452 1.003 0.912 0.069 -1.092 0.547  
## Blot\_16 148.869 146 0.418 1.013 1.117 0.141 0.918 0.296  
## Blot\_24 149.846 146 0.397 1.019 0.887 0.161 -1.125 0.535  
## Blot\_19 153.222 146 0.325 1.042 1.009 0.283 0.126 0.430  
## Blot\_30 168.153 146 0.101 1.144 1.185 1.011 2.248 0.283  
## Blot\_28 179.678 146 0.030 1.222 1.110 1.649 1.360 0.346  
## Blot\_25 183.566 146 0.019 1.249 1.070 1.300 0.791 0.359  
## Blot\_14 188.287 146 0.011 1.281 1.134 0.830 0.863 0.221  
## Blot\_13 193.280 146 0.005 1.315 1.162 2.002 1.971 0.311  
## Blot\_21 243.879 146 0.000 1.659 1.248 3.523 2.529 0.169  
## Blot\_22 236.313 146 0.000 1.608 0.888 1.313 -0.545 0.433  
## Blot\_31 224.002 146 0.000 1.524 1.065 2.097 0.650 0.359

1. Sort the table by increasing item discrimination values

print(item\_fit, sort\_by = "discrim")

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## Blot\_21 243.879 146 0.000 1.659 1.248 3.523 2.529 0.169  
## Blot\_14 188.287 146 0.011 1.281 1.134 0.830 0.863 0.221  
## Blot\_6 114.442 146 0.975 0.779 1.007 0.031 0.145 0.241  
## Blot\_30 168.153 146 0.101 1.144 1.185 1.011 2.248 0.283  
## Blot\_16 148.869 146 0.418 1.013 1.117 0.141 0.918 0.296  
## Blot\_33 134.675 146 0.739 0.916 1.079 -0.152 0.583 0.308  
## Blot\_13 193.280 146 0.005 1.315 1.162 2.002 1.971 0.311  
## Blot\_28 179.678 146 0.030 1.222 1.110 1.649 1.360 0.346  
## Blot\_25 183.566 146 0.019 1.249 1.070 1.300 0.791 0.359  
## Blot\_31 224.002 146 0.000 1.524 1.065 2.097 0.650 0.359  
## Blot\_5 108.213 146 0.992 0.736 0.956 -0.547 -0.184 0.377  
## Blot\_9 141.821 146 0.582 0.965 1.061 -0.085 0.614 0.379  
## Blot\_23 135.256 146 0.728 0.920 1.057 -0.336 0.615 0.383  
## Blot\_2 107.906 146 0.992 0.734 0.993 -0.686 0.012 0.395  
## Blot\_11 140.362 146 0.616 0.955 1.017 -0.131 0.200 0.412  
## Blot\_1 98.894 146 0.999 0.673 0.966 -0.858 -0.156 0.413  
## Blot\_7 93.946 146 1.000 0.639 0.958 -1.065 -0.228 0.420  
## Blot\_34 113.985 146 0.977 0.775 0.988 -0.685 -0.044 0.422  
## Blot\_4 127.534 146 0.862 0.868 0.992 -0.479 -0.034 0.426  
## Blot\_19 153.222 146 0.325 1.042 1.009 0.283 0.126 0.430  
## Blot\_22 236.313 146 0.000 1.608 0.888 1.313 -0.545 0.433  
## Blot\_3 132.504 146 0.781 0.901 0.978 -0.563 -0.235 0.465  
## Blot\_20 116.335 146 0.966 0.791 0.894 -0.444 -0.592 0.466  
## Blot\_29 103.054 146 0.997 0.701 0.924 -0.944 -0.512 0.467  
## Blot\_15 123.802 146 0.909 0.842 0.965 -1.089 -0.421 0.481  
## Blot\_35 106.324 146 0.994 0.723 0.915 -0.957 -0.630 0.496  
## Blot\_32 124.722 146 0.898 0.848 0.958 -1.103 -0.523 0.500  
## Blot\_10 99.791 146 0.999 0.679 0.907 -1.228 -0.742 0.502  
## Blot\_27 88.838 146 1.000 0.604 0.829 -1.004 -0.975 0.515  
## Blot\_18 107.506 146 0.993 0.731 0.893 -1.087 -0.935 0.518  
## Blot\_26 110.900 146 0.986 0.754 0.896 -1.605 -1.275 0.530  
## Blot\_24 149.846 146 0.397 1.019 0.887 0.161 -1.125 0.535  
## Blot\_8 147.396 146 0.452 1.003 0.912 0.069 -1.092 0.547  
## Blot\_17 109.310 146 0.990 0.744 0.865 -1.352 -1.475 0.567  
## Blot\_12 34.257 146 1.000 0.233 0.664 -1.598 -1.316 0.623

Now let’s print a summary of the estimated item difficulty parameters.

# Displaying item parameters, their standard errors and their 95% confidence intervals  
summary(fit\_rasch\_blot)

##   
## Results of RM estimation:   
##   
## Call: eRm::RM(X = blot\_items)   
##   
## Conditional log-likelihood: -1924.041   
## Number of iterations: 33   
## Number of parameters: 34   
##   
## Item (Category) Difficulty Parameters (eta): with 0.95 CI:  
## Estimate Std. Error lower CI upper CI  
## Blot\_2 -0.701 0.251 -1.193 -0.209  
## Blot\_3 0.741 0.192 0.365 1.117  
## Blot\_4 0.003 0.213 -0.414 0.421  
## Blot\_5 -0.983 0.272 -1.516 -0.449  
## Blot\_6 -2.433 0.459 -3.333 -1.533  
## Blot\_7 -0.637 0.247 -1.121 -0.153  
## Blot\_8 0.851 0.190 0.479 1.224  
## Blot\_9 0.183 0.206 -0.222 0.587  
## Blot\_10 -0.190 0.222 -0.624 0.244  
## Blot\_11 0.183 0.206 -0.222 0.587  
## Blot\_12 -1.766 0.354 -2.459 -1.073  
## Blot\_13 0.995 0.188 0.627 1.364  
## Blot\_14 -0.701 0.251 -1.193 -0.209  
## Blot\_15 0.995 0.188 0.627 1.364  
## Blot\_16 -0.293 0.227 -0.737 0.151  
## Blot\_17 0.392 0.200 0.001 0.784  
## Blot\_18 -0.044 0.215 -0.465 0.378  
## Blot\_19 0.473 0.198 0.085 0.860  
## Blot\_20 -0.836 0.261 -1.347 -0.325  
## Blot\_21 2.294 0.194 1.913 2.675  
## Blot\_22 -1.061 0.279 -1.607 -0.515  
## Blot\_23 0.351 0.201 -0.042 0.745  
## Blot\_24 0.226 0.205 -0.176 0.627  
## Blot\_25 0.512 0.197 0.126 0.897  
## Blot\_26 0.778 0.191 0.403 1.153  
## Blot\_27 -0.908 0.266 -1.429 -0.386  
## Blot\_28 1.622 0.186 1.257 1.987  
## Blot\_29 -0.458 0.236 -0.919 0.004  
## Blot\_30 1.066 0.187 0.699 1.434  
## Blot\_31 0.183 0.206 -0.222 0.587  
## Blot\_32 1.137 0.187 0.771 1.503  
## Blot\_33 -0.516 0.239 -0.984 -0.047  
## Blot\_34 -0.401 0.232 -0.857 0.054  
## Blot\_35 -0.293 0.227 -0.737 0.151  
##   
## Item Easiness Parameters (beta) with 0.95 CI:  
## Estimate Std. Error lower CI upper CI  
## beta Blot\_1 0.767 0.256 0.266 1.268  
## beta Blot\_2 0.701 0.251 0.209 1.193  
## beta Blot\_3 -0.741 0.192 -1.117 -0.365  
## beta Blot\_4 -0.003 0.213 -0.421 0.414  
## beta Blot\_5 0.983 0.272 0.449 1.516  
## beta Blot\_6 2.433 0.459 1.533 3.333  
## beta Blot\_7 0.637 0.247 0.153 1.121  
## beta Blot\_8 -0.851 0.190 -1.224 -0.479  
## beta Blot\_9 -0.183 0.206 -0.587 0.222  
## beta Blot\_10 0.190 0.222 -0.244 0.624  
## beta Blot\_11 -0.183 0.206 -0.587 0.222  
## beta Blot\_12 1.766 0.354 1.073 2.459  
## beta Blot\_13 -0.995 0.188 -1.364 -0.627  
## beta Blot\_14 0.701 0.251 0.209 1.193  
## beta Blot\_15 -0.995 0.188 -1.364 -0.627  
## beta Blot\_16 0.293 0.227 -0.151 0.737  
## beta Blot\_17 -0.392 0.200 -0.784 -0.001  
## beta Blot\_18 0.044 0.215 -0.378 0.465  
## beta Blot\_19 -0.473 0.198 -0.860 -0.085  
## beta Blot\_20 0.836 0.261 0.325 1.347  
## beta Blot\_21 -2.294 0.194 -2.675 -1.913  
## beta Blot\_22 1.061 0.279 0.515 1.607  
## beta Blot\_23 -0.351 0.201 -0.745 0.042  
## beta Blot\_24 -0.226 0.205 -0.627 0.176  
## beta Blot\_25 -0.512 0.197 -0.897 -0.126  
## beta Blot\_26 -0.778 0.191 -1.153 -0.403  
## beta Blot\_27 0.908 0.266 0.386 1.429  
## beta Blot\_28 -1.622 0.186 -1.987 -1.257  
## beta Blot\_29 0.458 0.236 -0.004 0.919  
## beta Blot\_30 -1.066 0.187 -1.434 -0.699  
## beta Blot\_31 -0.183 0.206 -0.587 0.222  
## beta Blot\_32 -1.137 0.187 -1.503 -0.771  
## beta Blot\_33 0.516 0.239 0.047 0.984  
## beta Blot\_34 0.401 0.232 -0.054 0.857  
## beta Blot\_35 0.293 0.227 -0.151 0.737

Notice that *eRm* not only reports item difficulty parameters (“Item (Category) Difficulty Parameters (eta)”) but also item easiness parameters (“Item Easiness Parameters (beta)”). These are just the difficulty parameters with reversed signs (e.g., difficulty of Blot\_35 = -0.293 becomes easiness of 0.293). This is for technical reasons and we can just ignore the easiness parameters.

Just like Winsteps, *eRm* also reports person separation reliability together with statistics of total person variance and the mean square measurement error.

# Person separation reliability, total person variance,   
# and model error variance  
summary(SepRel(pparameters\_blot))

## Separation Reliability: 0.8308  
##   
## Observed Variance: 1.8415 (Squared Standard Deviation)  
## Mean Square Measurement Error: 0.3116 (Model Error Variance)

# Plots

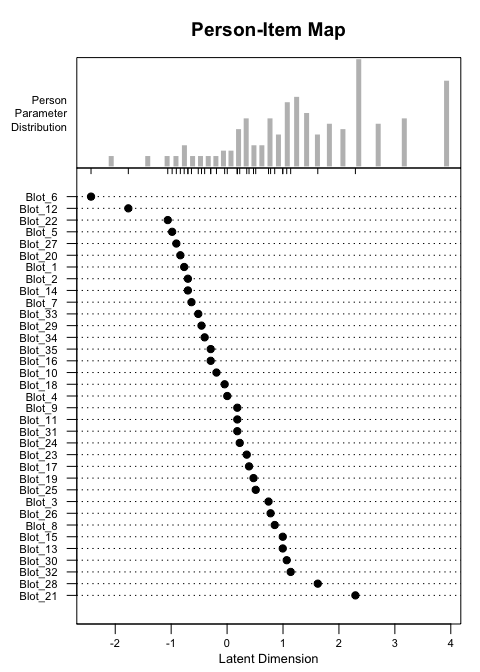
The following code lines generate *eRm* versions of the Wright and pathway map.

## Wright Map

The Wright map displays the location of item parameters and the distribution of person parameters along the latent dimension. These graphs are also referred to as person-item maps. The argument sort=TRUE orders the items in increasing difficulty to make the graph easier to read. The x-axis is the latent dimension of the Rasch model (what we are measuring). On the y-axis is each item. Each item gets a dot for it’s difficulty location on the latent dimension. The distribution of the person abilities are at the very top as a histogram - the height of the bar shows you how many people are at each ability score. The short tiny black bars beneath the person distribution correspond to the locations of the items.

We can use the plot to see how well the item difficulty distribution matches our person ability distribution. You can see that there are some persons whose abilities are much higher than the item difficulties.

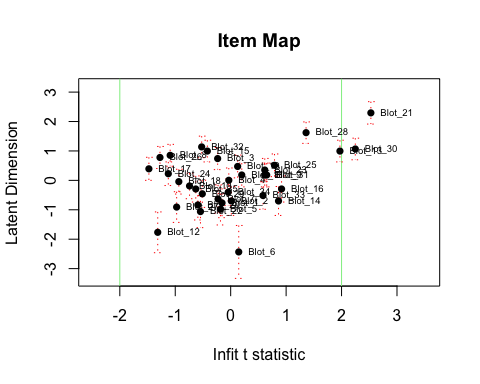
# Wright map  
plotPImap(fit\_rasch\_blot, irug = TRUE, sort = TRUE)



## Pathway Map

The pathway map displays the location of each item against its infit *t*-statistic. This graph lets us examine how well items fit the Rasch model at different values of the latent dimension. Items are more difficult as they move higher on the y-axis, and they are less predictable as they move to the right on the x-axis. Item 21 is both the most difficult item and has the least predictable pattern of responses.

# Pathway map with  
# 95% confidence intervals for the item parameters  
plotPWmap(fit\_rasch\_blot, itemCI = list())



Notice that the *eRm* version of the pathway map also prints 95% confidence intervals for the item parameters. The red dotted lines represent the confidence intervals for the item difficulty estimation. The broader the bands, the less precise our estimate of the item difficulty is.

The Rasch model expects all of our items to have an infit *t*-statistic of zero, but this is an impossible standard for the whole test. The green vertical lines at -2 and +2 roughly cut-off items that are not behaving as the model expects them to. Items outside of this cut-off are likely to be misfitting.

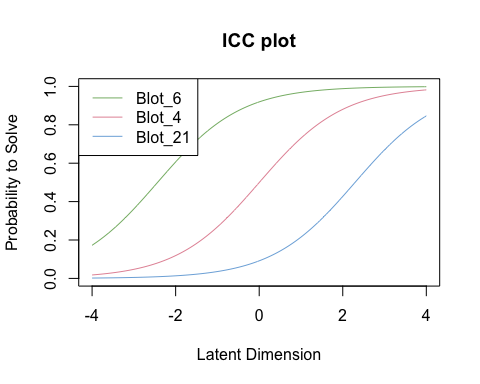
We can see that most of our items lie within the infit bounds, which is good. The items with the worst infit (items 13, 21, 30) are also some of the hardest items. Graphs like this can help us see such patterns.

In RStudio, you can navigate between both plots by clicking on the arrow keys of the “Plots” tab. If the plot does not look right, re-size the “Plots” pane and re-run the code. If you are having trouble with the plots or you receive a warning when you run the code, try clearing your old plots by clicking the broom icon and making the plot pane bigger. You can inspect a graph in more detail by clicking on the “Zoom” button. To save the graph, click on “Export” in the “Plots” tab. You can then save the file as an image (PNG, JPG, etc.) or as PDF. This is useful when you only want to share the graph or insert it into a document. Alternatively, you can copy it to the clipboard (option “Copy Plot to clipboard”) and then paste it directly into a document.

## Expected Score ICCs

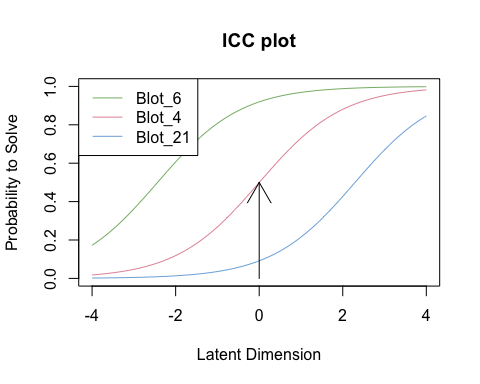
Expected score ICCs, also known as theoretical ICCs, show the probability of positive responses to an item based on the ability of a person. Under the Rasch model, each curve has an identical, monotonically increasing “S” shape. Because these are expected score curves, the only thing that differs between these items are their difficulties. The difficulty of the item determines where the inflection point (center) of the curve is located on the x-axis. If you drew a line down from the inflection point to the x-axis, it would be located exactly at that item’s difficulty.

# Expected Score ICCs – items # 6, 4, 21  
plotjointICC(fit\_rasch\_blot, item.subset = c("Blot\_4", "Blot\_6", "Blot\_21"))



By plotting the ICCs next to each other, we can see that item 6 is easier than item 4 because item 6’s curve is further to the left. Item 21 is the hardest item that was plotted. Referring back to the matrix of item difficulties above (you can run summary(fit\_rasch\_blot) to see these again), item 6 has a difficulty of -2.4, item 4’s is 0, and item 21’s is 2.3. If we drew a line down from the inflection point of each curve, we would find those exact difficulties. Item difficulties are also sometimes called “locations.” Below, we’ll show where item 4’s inflection point & difficulty is located at.

# Expected Score ICCs – items # 6, 4, 21  
plotjointICC(fit\_rasch\_blot, item.subset = c("Blot\_4", "Blot\_6", "Blot\_21"))  
  
# Indicate an inflection point & location of item 4's difficulty  
arrows(x0=0, x1=0, y0=0, y1=.5)

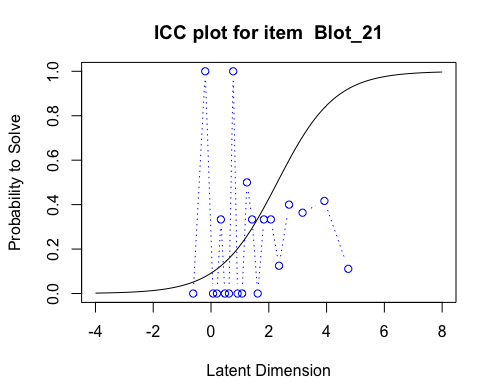


## Empirical vs Theoretical ICCs:

We can also plot the expected ICC against the empirical function, that is, the actual frequencies of positive responses across the latent dimension. *Empirical* means *observed*, so the empirical function is the actual response function that the item produces, not what the Rasch model assumes. The relative frequencies of the positive responses are calculated for each raw score group and plotted at the position of the corresponding person parameter. This is helpful because it can show us if the Rasch model is accurately predicting how people are responding. We can see how many people actually got an item correct across the range of abilities. As mentioned before, we do not expect the data to perfectly fit the model, but we want to see that the two ICCs follow the same trend.

If the item fits the Rasch model, the empirical function (below, in blue) should roughly mirror the theoretical ICC (in black). For our empirical function, each dot is a raw score category. The placement of the dot on the y-axis shows how many people at that raw score actually got that item correct. Here for item 21, we have a relatively poor match between the expected and the empirical functions. We can see that there is a large difference between our blue and black lines, especially toward the the right side of the x-axis. Our participants with the highest ability (or “latent dimension”) are relatively unlikely to get item 21 correct, which was not what the Rasch model predicted. This reflects what we saw in item 21’s fit statistics above.

# Actual vs theoretical ICC:  
plotICC(fit\_rasch\_blot, item.subset = c("Blot\_21"), empICC = list("raw", type = "b", col = "blue", lty = "dotted"), xlim = c(-4,8))

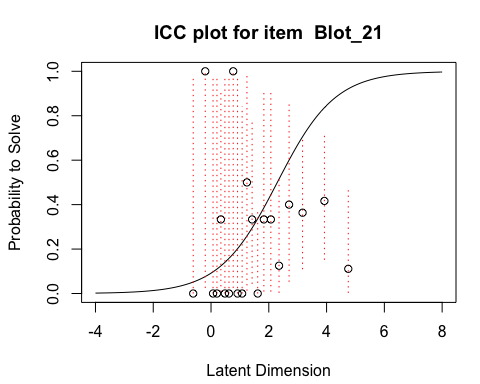


But how do we know how trustworthy these results are? We can also repeat the same graph along with the 95% confidence intervals (CIs) for the empirical ICC. The red dotted lines show the 95% CI of the estimated probability to solve at each raw score. When the bands are large, it means we have a less precise estimate of this probability. In addition, when the red dotted lines do not cross the black line, it means that it is unlikely that our empirical probabilities are matching what the Rasch model expects.

The 95% CIs can be very helpful in interpreting the mismatch between our empirical and theoretical curves. We can see that at the lower ends of the latent dimension, the vertical red lines stretch from 0 (0%) to 1 (100%) for each dot. This is because we only had one or two people in our sample with that raw score. With one person, we can’t be very sure what the true probability of responding is. Correspondingly, the confidence interval stretches from 0% to 100% - the true probability of getting the item correct could be anywhere in that interval. Our sample size is too small to accurately estimate these probabilities at some raw scores.

When we look at the red 95% confidence intervals below, we can see that they are generally bigger towards the lower end of the latent dimension where we have less people in our sample. The CI bands are smaller towards the higher end of the latent dimension where we have more people in our sample. Where the CI bands are big and overlap with the theoretical black curve, we can’t be sure that our empirical and theoretical probabilities mismatch. On the other hand, at higher end of the latent dimension the bands are smaller and do not overlap with the black theoretical curve. Since the CI’s do not cross the black theoretical line, it is likely that our high scorers are not responding as they should to this question.

# Actual vs theoretical ICC, with 95% confidence intervals:  
plotICC(fit\_rasch\_blot, item.subset = c("Blot\_21"), empICC = list("raw"), empCI = list(), xlim = c(-4,8))



#### References

Cureton, E. E. (1966). Corrected item-test correlations. *Psychometrika, 31*, 93-96