Open Tests: Harvard Measurement Lab

Emma Dwight

6/23/2020

In this tutorial we create and score a test for MCAS 4th grade ELA, using simulated student responses, and released questions from 2018.

Ingredients:

1) Released test questions (items) to create the test, like these: https://mcas.digitalitemlibrary.com/home?subject=ELA&grades=Grade%204&view=ALL
2) Released item IRT parameters, like table M5 here: http://www.mcasservicecenter.com/documents/MA/Technical%20Report/2018/NextGen/Appendix%20M%20-%20Plots%20and%20IRT%20Parameters.pdf Example file saved as "tablem5.xlsx"
3) Common item numbers/identifiers that relate each question (item) to its parameters: currently missing for MCAS
4) Student responses for each of the test questions, graded as correct/incorrect: we simulate fake data below (Note: we plan to add code that could estimate student ability using only sum-scores, rather than full-pattern scores)
5) A table that converts theta scores to scale scores (and possibly also achievement levels), like table N2, here: http://www.mcasservicecenter.com/documents/MA/Technical%20Report/2018/NextGen/Appendix%20N%20-%20Scaled%20Score%20Distributions%20and%20Look-up%20Tables_4.17.19.pdf Example file saved as "tablen2.xlsx"

Progress:

Implemented so far: - Import 3PL item parameters - Import theta to scale score table - Simulate student responses, as full-pattern 0/1 scores - Estimate student ability on theta scale - Convert theta scores to scale scores - Estimate achievement levels for cutoffs known/given for scale scores - Export student ability data, including thetas, scale scores, and achievement levels - Report Classical Test Theory statistics - Plot Item Characteristic Curves and Test Characteristic Curve - Plot Item Information Functions and Test Information Function

To do next: - Report standard errors on scale scores if possible?

Cool, harder to-dos: - How should I think about standard errors on theta scale from ability estimation and the scale-score standard errors MCAS released? How should I report and explain standard errors? - Implement sum scores -> scale scores methodology (estimation, probably sans standard errors) - Add support for polytomously scored items? (Hand-scored, GRM stuff for MCAS)

Future extras/usability extensions: - Extend functionality for 1PL and 2PL parameters, if that's how a different state does things (possibly just zero out the c column) - Add code to import student responses, including student names/identifiers to attach to the thetas, scale scores, and - Add code to dichotomously score multi-choice questions if a key is provided (functions already exist for this, eg. irtoys::sco)

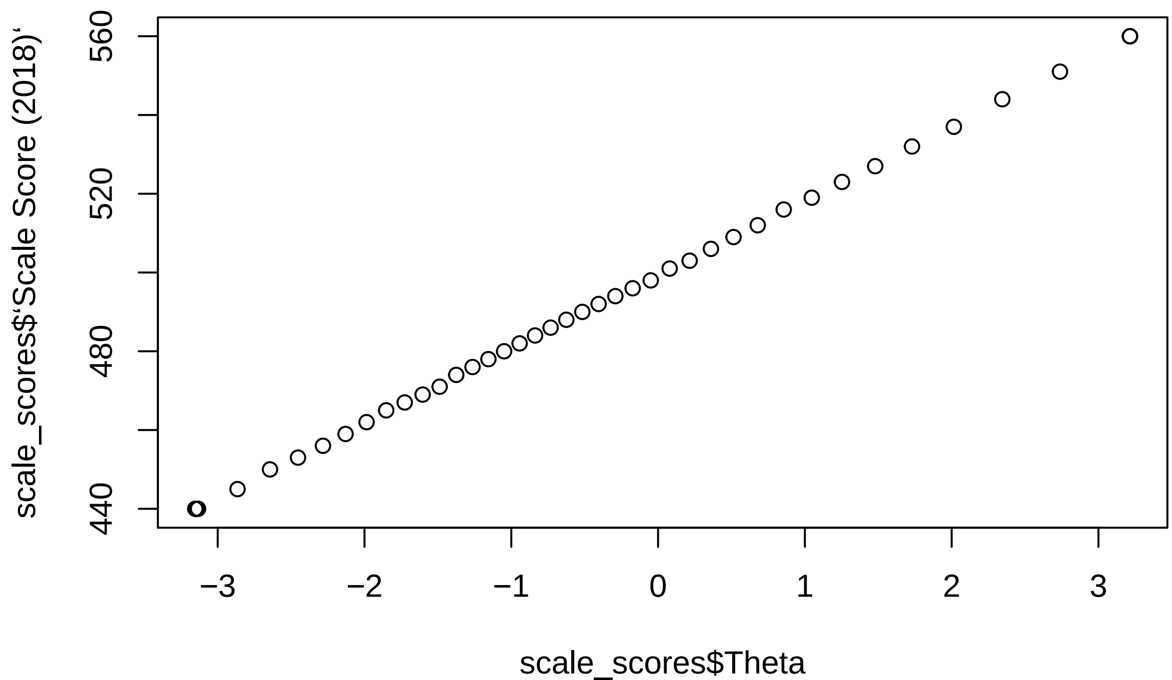
```r
library(readxl)

# Import item parameter data
item_parameters_raw <- read_excel("tablem5.xlsx")
# Separate the item parameters from the standard errors
my_ip <- as.matrix(item_parameters_raw  %>% dplyr::select(a, b, c))
my_se <- as.matrix(item_parameters_raw  %>% dplyr::select("se(a)", "se(b)", "se(c)"))

# Import student responses
# Note: use the "upload" function next to the file menu in the window on the right hand side of the con

# Import theta -> scale score table
scale_scores_raw <- read_excel("tablen2.xlsx")
# Pull out the relevant columns:
scale_scores <- scale_scores_raw  %>% dplyr::select("Theta", "Scale Score (2018)")

# Plot relationship between thetas and scale scores:
plot(x = scale_scores$Theta, y = scale_scores$`Scale Score (2018)`)
```




```r
# Relationship is a straight line! Learn the model to be able to convert later.
theta_to_scale_score <- lm(`Scale Score (2018)` ~ Theta, data = scale_scores)
summary(theta_to_scale_score)
```

```
##
## Call:
## lm(formula = `Scale Score (2018)` ~ Theta, data = scale_scores)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48284 -0.21294 -0.04279  0.35221  0.49858
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 499.42554    0.04645 10751.0   <2e-16 ***
## Theta        18.85391    0.02377   793.3   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2994 on 43 degrees of freedom
## Multiple R-squared:  0.9999, Adjusted R-squared:  0.9999
## F-statistic: 6.294e+05 on 1 and 43 DF,  p-value: < 2.2e-16
```

```r
# Load IRT library
library(irtoys)
```

```
## Loading required package: sm

## Warning in fun(libname, pkgname): couldn't connect to display ":0"

## Package 'sm', version 2.2-5.6: type help(sm) for summary information

## Loading required package: ltm

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:sm':
##
##     muscle

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: msm

## Loading required package: polycor
```

```r
# Use parameter estimates to simulate answers for 100 students
set.seed(88)
sim_thetas <- rnorm(100)
sim_responses <- sim(my_ip, sim_thetas)

# Put the parameter estimates and standard errors into the list structure that irtoys functions expect
# Note: ability estimation function does not need standard errors to run
parameter_list <- list(est = my_ip, se = my_se, vcm = NA)

# Estimate student thetas, based on full-pattern scoring, using MLE (several other methods exist in thi
mod_MLE<- ability(resp = sim_responses, ip = parameter_list, method = "MLE")
mod_MLE
```

```
##                 est       sem  n
##   [1,]  1.19463657 1.0862808 15
##   [2,]  1.37310053 1.1410868 15
##   [3,]  3.99991978 2.3626913 15
##   [4,] -0.79221784 0.7661342 15
##   [5,] -0.45505265 0.7780056 15
##   [6,]  0.67182377 0.9478088 15
```

```
##   [7,]  2.16511825 1.4271091 15
##   [8,] -2.60827667 1.0334632 15
##   [9,] -2.45136188 0.9871825 15
##  [10,]  1.15067729 1.0733509 15
##  [11,] -0.71719368 0.7671307 15
##  [12,]  1.73279682 1.2624897 15
##  [13,]  1.40842617 1.1523689 15
##  [14,] -1.08433633 0.7712821 15
##  [15,] -0.22458413 0.7969156 15
##  [16,]  0.43538827 0.8967711 15
##  [17,] -0.02442786 0.8202295 15
##  [18,]  0.46985620 0.9037363 15
##  [19,] -0.96907592 0.7675345 15
##  [20,]  0.66346828 0.9458771 15
##  [21,]  3.24590128 1.9286394 15
##  [22,] -0.17430406 0.8021786 15
##  [23,] -0.11290465 0.8091483 15
##  [24,] -0.98883114 0.7680181 15
##  [25,] -0.36629761 0.7842653 15
##  [26,] -1.23205426 0.7793474 15
##  [27,]  0.08111884 0.8350259 15
##  [28,] -0.41975064 0.7803407 15
##  [29,]  0.17984639 0.8503896 15
##  [30,] -0.82495561 0.7659960 15
##  [31,] -1.65285045 0.8223066 15
##  [32,]  0.88270743 0.9995634 15
##  [33,] -1.28366278 0.7830267 15
##  [34,]  0.09699870 0.8373986 15
##  [35,]  1.09902944 1.0584524 15
##  [36,] -1.29504600 0.7838981 15
##  [37,] -1.29281478 0.7837256 15
##  [38,]  0.66307464 0.9457863 15
##  [39,] -1.09187809 0.7716052 15
##  [40,]  2.00415683 1.3634616 15
##  [41,]  1.09894867 1.0584294 15
##  [42,]  3.99991978 2.3626913 15
##  [43,] -0.98365611 0.7678851 15
##  [44,]  3.50484089 2.0692361 15
##  [45,] -0.20828185 0.7985779 15
##  [46,] -0.09241462 0.8116061 15
##  [47,] -0.10909455 0.8096004 15
##  [48,]  1.19463657 1.0862808 15
##  [49,]  1.01813122 1.0357597 15
##  [50,] -0.14577724 0.8053429 15
##  [51,]  1.12499595 1.0659031 15
##  [52,]  0.70986468 0.9567197 15
##  [53,] -2.38398252 0.9687941 15
##  [54,] -0.22947329 0.7964254 15
##  [55,] -1.19919355 0.7772370 15
##  [56,] -0.99837533 0.7682753 15
##  [57,] -1.19533765 0.7770012 15
##  [58,]  0.91025941 1.0067433 15
##  [59,]  0.66570705 0.9463938 15
##  [60,] -1.05814150 0.7702342 15
```

```
##  [61,]  1.12175398 1.0649685 15
##  [62,] -1.59242904 0.8143217 15
##  [63,] -0.13484125 0.8065900 15
##  [64,]  0.54103137 0.9186378 15
##  [65,] -0.59821214 0.7706474 15
##  [66,]  0.11452882 0.8400618 15
##  [67,] -0.88502156 0.7662120 15
##  [68,]  1.87700692 1.3151576 15
##  [69,]  0.48020898 0.9058605 15
##  [70,]  1.01243947 1.0341929 15
##  [71,]  0.97826330 1.0248687 15
##  [72,]  0.11452882 0.8400618 15
##  [73,]  3.35419651 1.9864123 15
##  [74,]  0.97839718 1.0249050 15
##  [75,]  0.83420751 0.9871565 15
##  [76,]  0.49885350 0.9097234 15
##  [77,]  0.42984335 0.8956661 15
##  [78,]  0.93238595 1.0125779 15
##  [79,] -1.65108281 0.8220644 15
##  [80,] -0.50917684 0.7748243 15
##  [81,] -0.74401718 0.7666658 15
##  [82,]  0.25609195 0.8632424 15
##  [83,]  0.56892697 0.9246667 15
##  [84,] -0.22517170 0.7968565 15
##  [85,]  2.95762823 1.7818074 15
##  [86,] -0.24650170 0.7947477 15
##  [87,] -0.27538289 0.7920085 15
##  [88,]  0.68528603 0.9509406 15
##  [89,] -2.55907600 1.0184228 15
##  [90,] -0.50489863 0.7750581 15
##  [91,]  0.02399325 0.8268064 15
##  [92,] -1.40059295 0.7930082 15
##  [93,] -2.61380286 1.0351832 15
##  [94,] -1.35739712 0.7890552 15
##  [95,]  3.99991978 2.3626913 15
##  [96,] -0.48769106 0.7760292 15
##  [97,]  0.73323865 0.9622888 15
##  [98,] -0.36367451 0.7844699 15
##  [99,]  2.16511825 1.4271091 15
## [100,] -0.77982267 0.7662336 15
```

```r
# Convert these estimated thetas to scale scores and achievement levels
ability_df <- as.data.frame(mod_MLE)
colnames(ability_df) <- c("Theta", "se(Theta)", "n")
ability_df$ScaleScore <- predict(theta_to_scale_score, newdata = ability_df)

ability_df$AchievementLevel <- 1
ability_df$AchievementLevel[ability_df$ScaleScore > 470] <- 2
ability_df$AchievementLevel[ability_df$ScaleScore > 500] <- 3
ability_df$AchievementLevel[ability_df$ScaleScore > 530] <- 4

# Look at the data
ability_df
```

```
##           Theta se(Theta)  n ScaleScore AchievementLevel
```

```
## 1    1.19463657 1.0862808 15   521.9491                 3
## 2    1.37310053 1.1410868 15   525.3138                 3
## 3    3.99991978 2.3626913 15   574.8396                 4
## 4   -0.79221784 0.7661342 15   484.4891                 2
## 5   -0.45505265 0.7780056 15   490.8460                 2
## 6    0.67182377 0.9478088 15   512.0920                 3
## 7    2.16511825 1.4271091 15   540.2465                 4
## 8   -2.60827667 1.0334632 15   450.2493                 1
## 9   -2.45136188 0.9871825 15   453.2078                 1
## 10   1.15067729 1.0733509 15   521.1203                 3
## 11  -0.71719368 0.7671307 15   485.9036                 2
## 12   1.73279682 1.2624897 15   532.0955                 4
## 13   1.40842617 1.1523689 15   525.9799                 3
## 14  -1.08433633 0.7712821 15   478.9816                 2
## 15  -0.22458413 0.7969156 15   495.1912                 2
## 16   0.43538827 0.8967711 15   507.6343                 3
## 17  -0.02442786 0.8202295 15   498.9650                 2
## 18   0.46985620 0.9037363 15   508.2842                 3
## 19  -0.96907592 0.7675345 15   481.1547                 2
## 20   0.66346828 0.9458771 15   511.9345                 3
## 21   3.24590128 1.9286394 15   560.6235                 4
## 22  -0.17430406 0.8021786 15   496.1392                 2
## 23  -0.11290465 0.8091483 15   497.2968                 2
## 24  -0.98883114 0.7680181 15   480.7822                 2
## 25  -0.36629761 0.7842653 15   492.5194                 2
## 26  -1.23205426 0.7793474 15   476.1965                 2
## 27   0.08111884 0.8350259 15   500.9549                 3
## 28  -0.41975064 0.7803407 15   491.5116                 2
## 29   0.17984639 0.8503896 15   502.8163                 3
## 30  -0.82495561 0.7659960 15   483.8719                 2
## 31  -1.65285045 0.8223066 15   468.2629                 1
## 32   0.88270743 0.9995634 15   516.0680                 3
## 33  -1.28366278 0.7830267 15   475.2235                 2
## 34   0.09699870 0.8373986 15   501.2543                 3
## 35   1.09902944 1.0584524 15   520.1465                 3
## 36  -1.29504600 0.7838981 15   475.0089                 2
## 37  -1.29281478 0.7837256 15   475.0509                 2
## 38   0.66307464 0.9457863 15   511.9271                 3
## 39  -1.09187809 0.7716052 15   478.8394                 2
## 40   2.00415683 1.3634616 15   537.2117                 4
## 41   1.09894867 1.0584294 15   520.1450                 3
## 42   3.99991978 2.3626913 15   574.8396                 4
## 43  -0.98365611 0.7678851 15   480.8798                 2
## 44   3.50484089 2.0692361 15   565.5055                 4
## 45  -0.20828185 0.7985779 15   495.4986                 2
## 46  -0.09241462 0.8116061 15   497.6832                 2
## 47  -0.10909455 0.8096004 15   497.3687                 2
## 48   1.19463657 1.0862808 15   521.9491                 3
## 49   1.01813122 1.0357597 15   518.6213                 3
## 50  -0.14577724 0.8053429 15   496.6771                 2
## 51   1.12499595 1.0659031 15   520.6361                 3
## 52   0.70986468 0.9567197 15   512.8093                 3
## 53  -2.38398252 0.9687941 15   454.4782                 1
## 54  -0.22947329 0.7964254 15   495.0991                 2
```

```
## 55  -1.19919355 0.7772370 15   476.8161              2
## 56  -0.99837533 0.7682753 15   480.6023              2
## 57  -1.19533765 0.7770012 15   476.8888              2
## 58   0.91025941 1.0067433 15   516.5875              3
## 59   0.66570705 0.9463938 15   511.9767              3
## 60  -1.05814150 0.7702342 15   479.4754              2
## 61   1.12175398 1.0649685 15   520.5750              3
## 62  -1.59242904 0.8143217 15   469.4020              1
## 63  -0.13484125 0.8065900 15   496.8833              2
## 64   0.54103137 0.9186378 15   509.6261              3
## 65  -0.59821214 0.7706474 15   488.1469              2
## 66   0.11452882 0.8400618 15   501.5849              3
## 67  -0.88502156 0.7662120 15   482.7394              2
## 68   1.87700692 1.3151576 15   534.8144              4
## 69   0.48020898 0.9058605 15   508.4794              3
## 70   1.01243947 1.0341929 15   518.5140              3
## 71   0.97826330 1.0248687 15   517.8696              3
## 72   0.11452882 0.8400618 15   501.5849              3
## 73   3.35419651 1.9864123 15   562.6652              4
## 74   0.97839718 1.0249050 15   517.8721              3
## 75   0.83420751 0.9871565 15   515.1536              3
## 76   0.49885350 0.9097234 15   508.8309              3
## 77   0.42984335 0.8956661 15   507.5298              3
## 78   0.93238595 1.0125779 15   517.0047              3
## 79  -1.65108281 0.8220644 15   468.2962              1
## 80  -0.50917684 0.7748243 15   489.8256              2
## 81  -0.74401718 0.7666658 15   485.3979              2
## 82   0.25609195 0.8632424 15   504.2539              3
## 83   0.56892697 0.9246667 15   510.1520              3
## 84  -0.22517170 0.7968565 15   495.1802              2
## 85   2.95762823 1.7818074 15   555.1884              4
## 86  -0.24650170 0.7947477 15   494.7780              2
## 87  -0.27538289 0.7920085 15   494.2335              2
## 88   0.68528603 0.9509406 15   512.3459              3
## 89  -2.55907600 1.0184228 15   451.1770              1
## 90  -0.50489863 0.7750581 15   489.9062              2
## 91   0.02399325 0.8268064 15   499.8779              2
## 92  -1.40059295 0.7930082 15   473.0189              2
## 93  -2.61380286 1.0351832 15   450.1451              1
## 94  -1.35739712 0.7890552 15   473.8333              2
## 95   3.99991978 2.3626913 15   574.8396              4
## 96  -0.48769106 0.7760292 15   490.2307              2
## 97   0.73323865 0.9622888 15   513.2499              3
## 98  -0.36367451 0.7844699 15   492.5689              2
## 99   2.16511825 1.4271091 15   540.2465              4
## 100 -0.77982267 0.7662336 15   484.7228              2
```

```r
# Round the scale scores to 0 dp before reporting
ability_df$ScaleScore <- round(ability_df$ScaleScore, 0)

# Export the data
write.csv(ability_df, file = "EstimatedScores.csv", row.names = F)
# The above file will appear in the file window on the bottom-right hand side of the screen
# To download to local computer, select this file using the checkbox, then use More...Export...
```

Diagnostics:

```
# Classical Test Theory EDA metrics:
# Note: Because the simulated data is pre-graded, we're saying the "answer key" is 1, 1, 1, 1, 1....
ctt <- tia(sim_responses, key = rep(1, 15))
ctt$testlevel$alpha
```

```
## [1] 0.476776
```

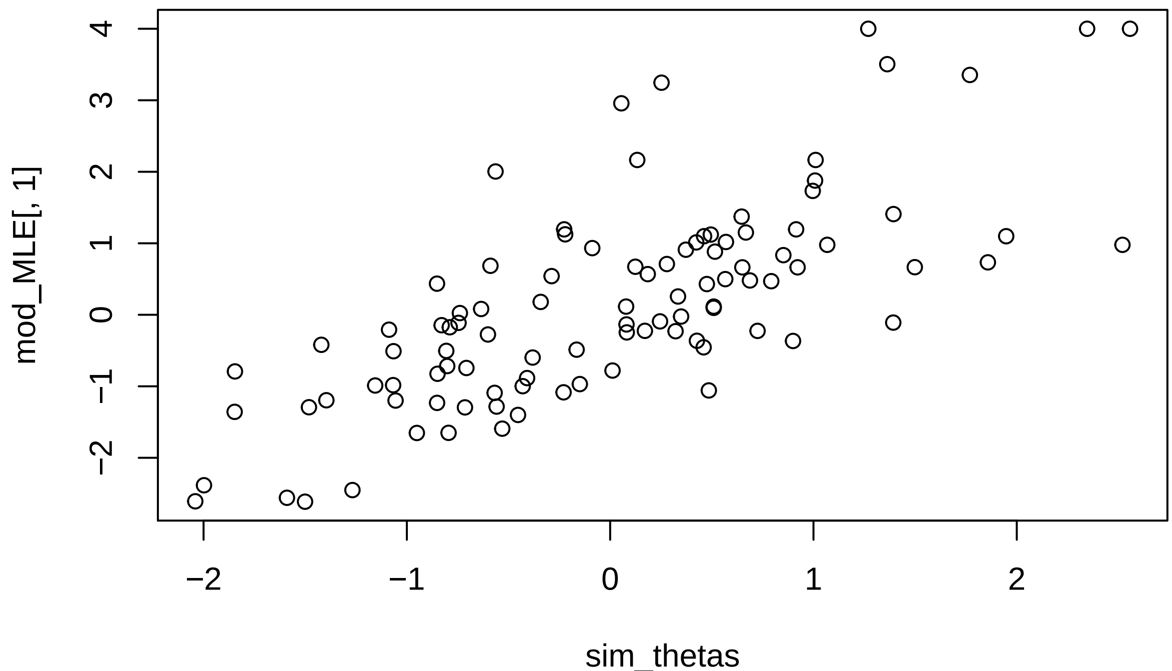
```
ctt$itemlevel
```

```
##        Prop. correct Item-sum cor. Alpha without
## Item1           0.70     0.3062258     0.4161638
## Item2           0.56     0.3015503     0.4131231
## Item3           0.59     0.3916676     0.3820999
## Item4           0.73     0.3936733     0.3912438
## Item5           0.52     0.3779185     0.3856171
## Item6           0.72     0.2879160     0.4227021
## Item7           0.67     0.3879720     0.3878075
## Item8           0.62     0.4152677     0.3748319
## Item9           0.83     0.3633950     0.4126288
## Item10          0.66     0.4775707     0.3546412
## Item11          0.76     0.3006990     0.4215429
## Item12          0.56     0.1816155     0.4513699
## Item13          0.78     0.3243888     0.4166970
## Item14          0.70     0.3619032     0.3986987
## Item15          0.90     0.3685442     0.4248155
```

```
# How does the MLE ability estimation perform compared to the "true thetas" from our simulation?
cor(mod_MLE[,1], sim_thetas)
```

```
## [1] 0.733215
```

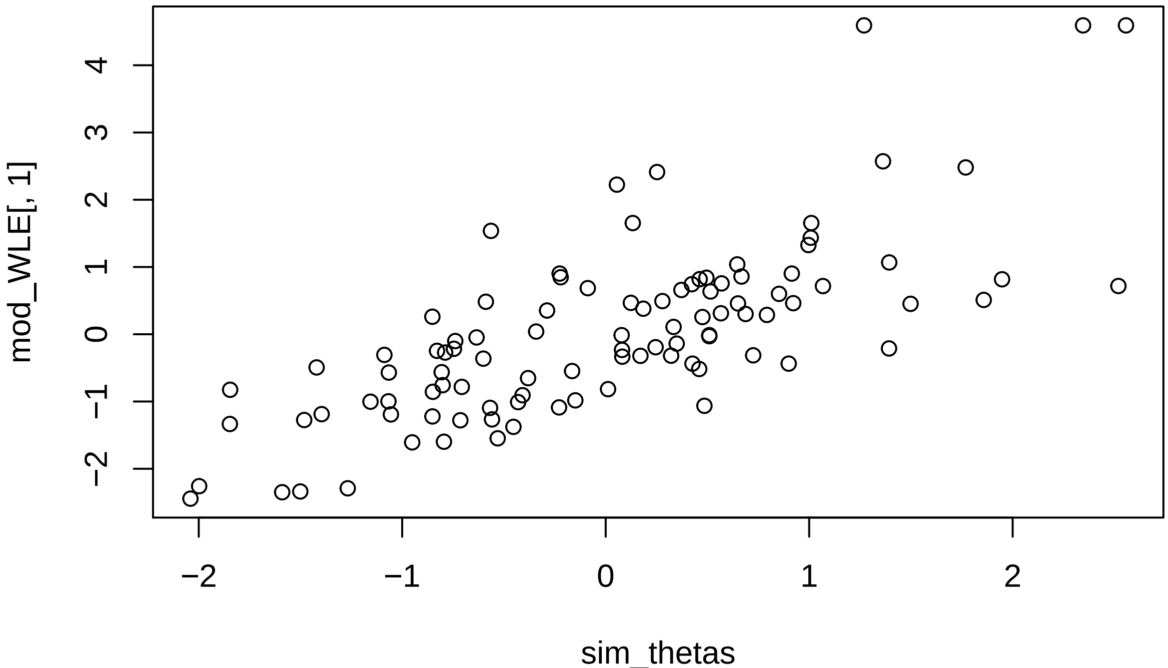
```
plot(sim_thetas, mod_MLE[,1])
```



```r
# How do other ability estimation methods perform?
mod_WLE <- ability(resp = sim_responses, ip = parameter_list, method = "WLE")
cor(mod_WLE[,1], sim_thetas)
```

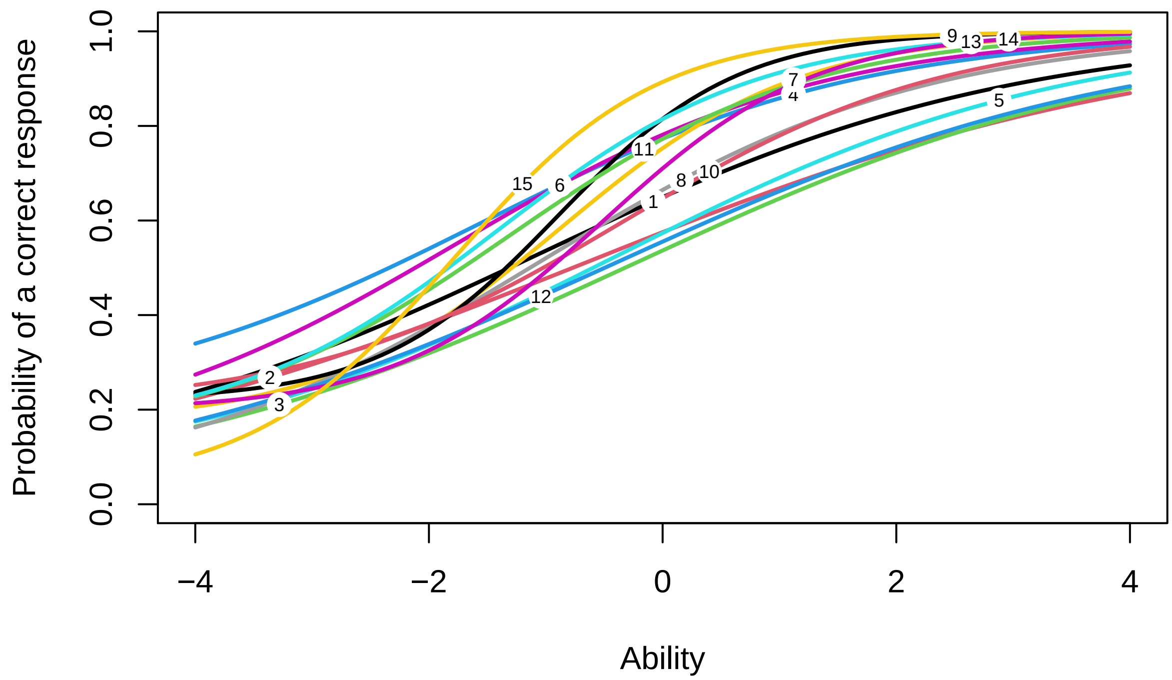
```
## [1] 0.7343091
```

```r
plot(sim_thetas, mod_WLE[,1])
```



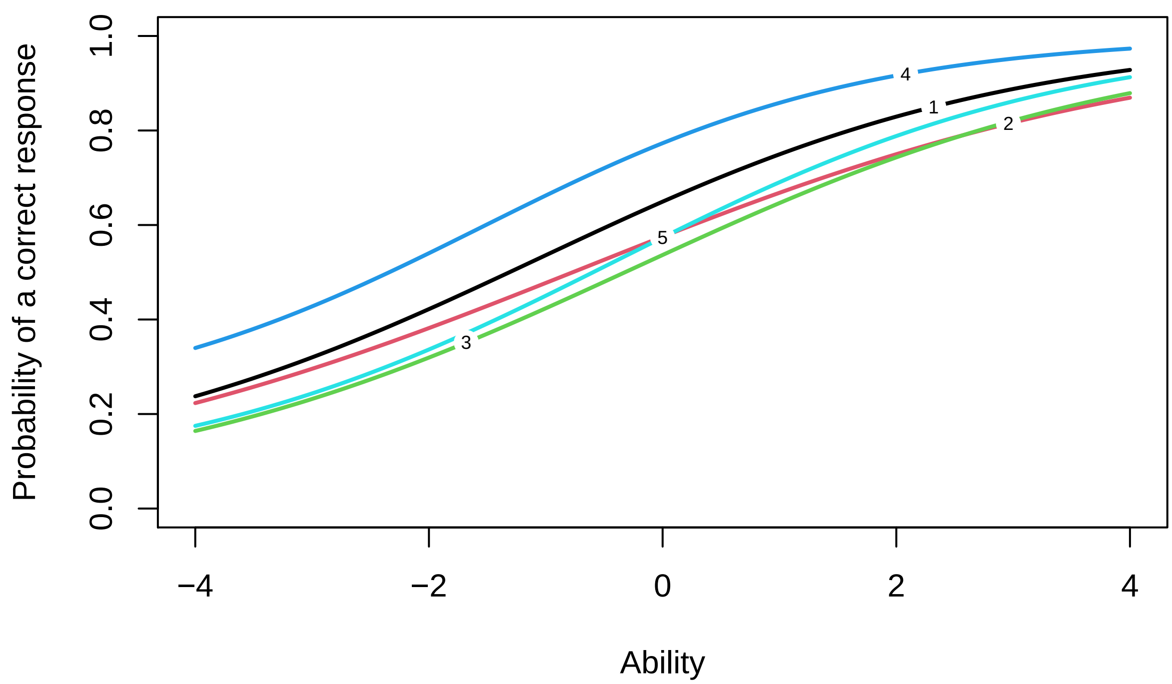

```r
# Item Characteristic Curves (this package calls them "item response functions")
plot(irf(my_ip), co = NA, label = T)
```

Item response function



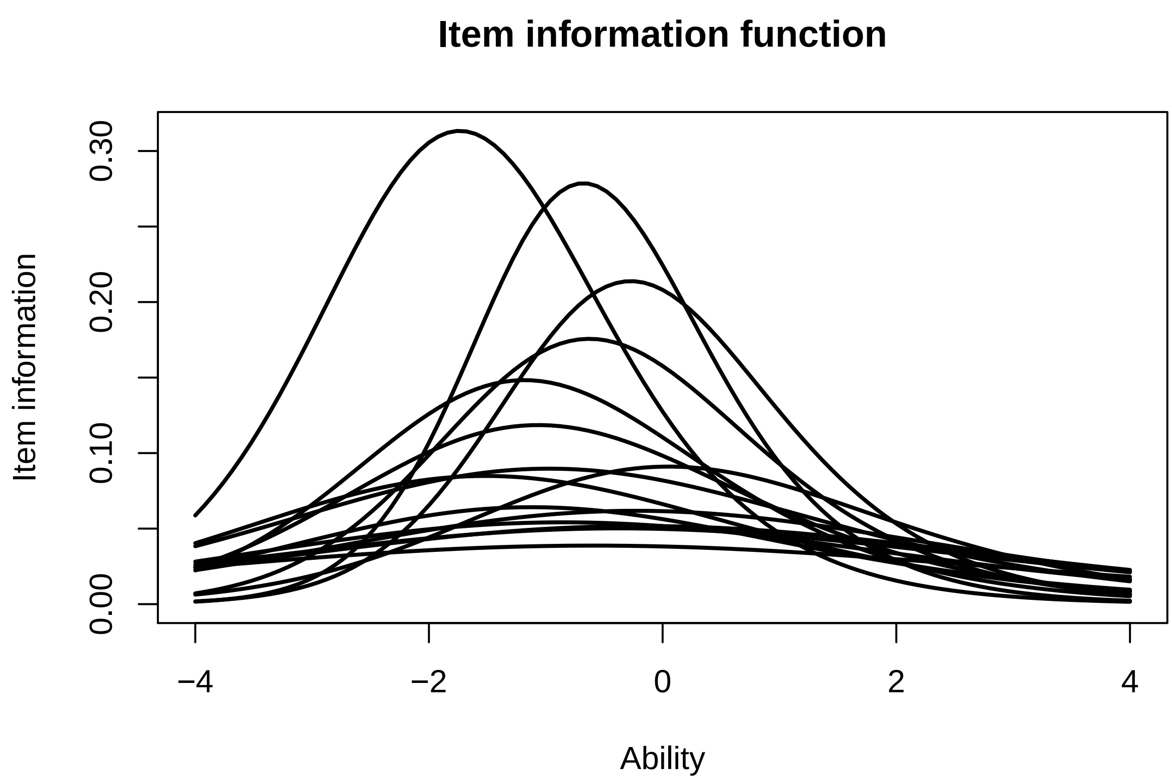

```
plot(irf(my_ip, items = 1:5), co = NA, label = T)
```

Item response function

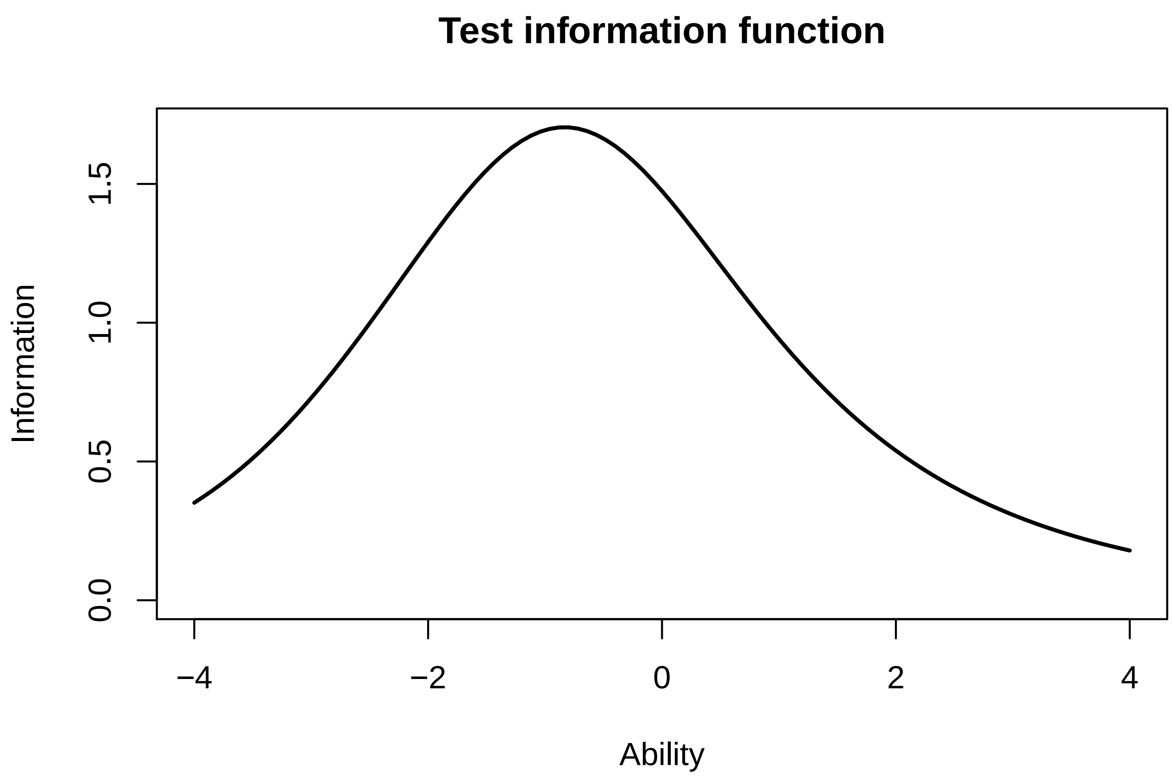



```
# Test Characteristic Curve (this package calls this a "test response function")
plot(trf(my_ip))
```

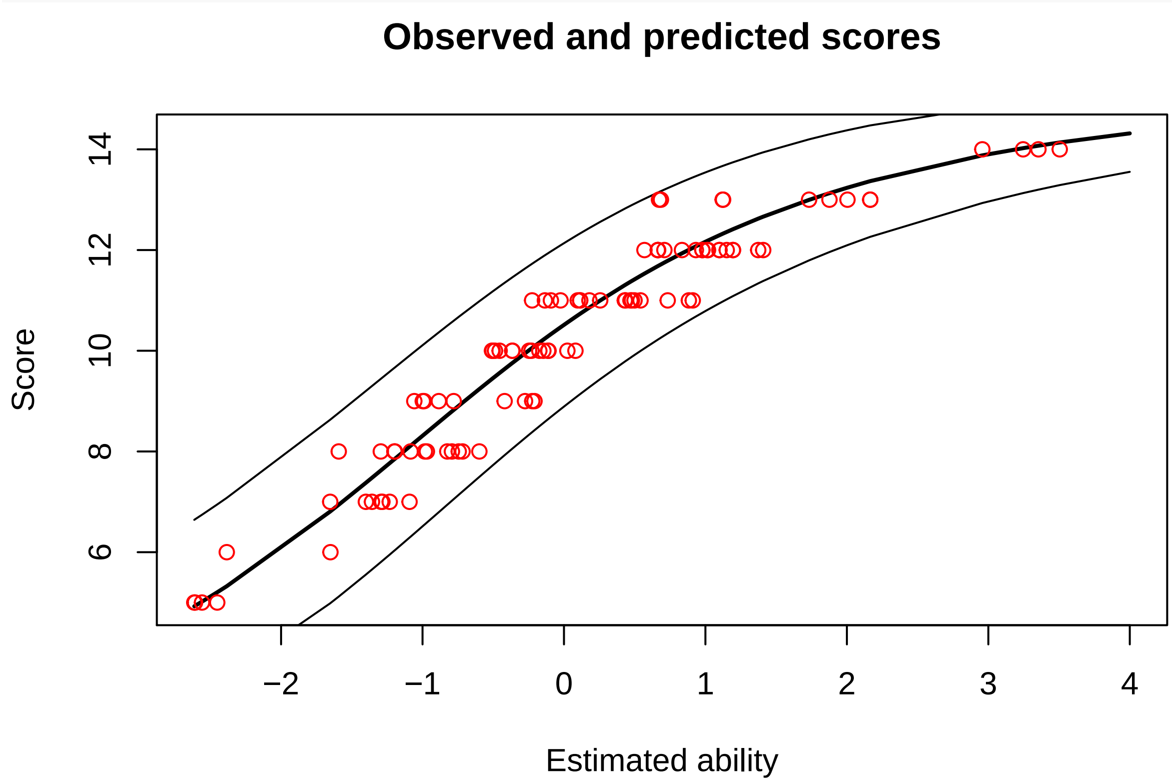

```r
# Overlaid item information curves
plot(iif(my_ip))
```



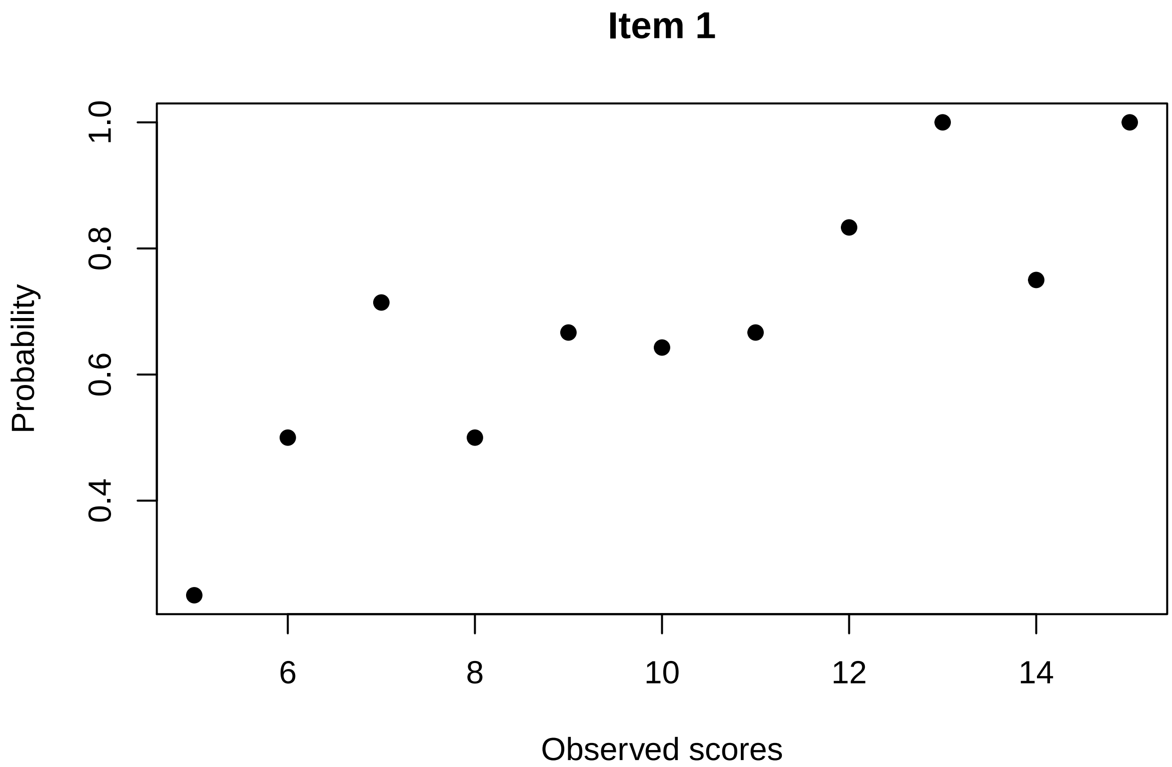

```r
# Test information function
plot(tif(my_ip))
```




```r
# Cool plot of observed sum scores and predicted sum scores against estimated ability, with +/- 1se ban
scp(sim_responses, my_ip)
```




```r
# "Empirical response function" for a selected item: observed sum scores vs. percent correct on this qu
erf(sim_responses, 1)
```




```
##      x         y
## 5    5 0.2500000
## 6    6 0.5000000
## 7    7 0.7142857
## 8    8 0.5000000
## 9    9 0.6666667
## 10  10 0.6428571
## 11  11 0.6666667
## 12  12 0.8333333
## 13  13 1.0000000
## 14  14 0.7500000
## 15  15 1.0000000
```