MiscPsycho

An R Package for Miscellaneous Psychometric Analyses

Harold C. Doran
American Institutes for Research (AIR)
hdoran@air.org

January 23, 2008

Contents

1 Introduction and Purpose

The purpose of the **MiscPsycho** package is to provide psychometricians with functions to analyze their data, including classical item analyses, item response models (IRT), and various functions commonly used in psychometrics. The functions provided in this package are intended to provide the user with psychometric procedures commonly used in testing programs. This document outlines the mathematical procedures used in the **MiscPsycho** package in `R`. Additionally, I provide examples of how to use these functions.

2 The Rasch Model

The Rasch, or 1-parameter logistic model, is an IRT model that assumes items can be adequately characterized via a single location (difficulty) parameter. Slopes across items (discrimination) and/or guessing are assumed to be constant across all items such that $a_j = a$ and $c_j = 0$ where a_j is the item discrimination parameter for item j and c_j is the guessing parameter (i.e., lower asymptote) for item j.

The basic Rasch model characterizes the probability of a correct response as:

$$\text{Prob}(X_{ij} = 1|\theta_i, \beta_j) = \frac{1}{1 + e^{-(\theta_i - \beta_j)}} \quad i = (1, \ldots, K); j = (1, \ldots, N) \tag{1}$$

where θ_i is the ability estimate of person i and β_j is the location parameter for item j.

3 Local Independence

The term *local independence* is commonly used in IRT. This simply means that a persons response to item j is independent of their response to any other item conditional on their ability. This assumption provides a convenient mathematical way to express the likelihood function since the joint density is then the product of the individual densities. Because the item responses are dichotomous, the data are assumed to following a Bernoulli distribution, thus giving rise to the following likelihood function:

$$L = \prod \text{Prob}(X_{ij} = 1|\theta_i, \beta_j)^{x_{ij}} \left[1 - \text{Prob}(X_{ij} = 1|\theta_i, \beta_j)\right]^{(1-x_{ij})} \tag{2}$$

where x_{ij} is the response of person i to item j such that:

$$x_{ij} = \begin{cases} 1 \text{ if correct response} \\ 0 \text{ otherwise} \end{cases}$$

The derivatives below are obtained from the log-likelihood:

$$\ln L = \sum x_{ij} \ln \left[\text{Prob}(X_{ij} = 1|\theta_i, \beta_j)\right] + (1 - x_{ij}) \ln \left[1 - \text{Prob}(X_{ij} = 1|\theta_i, \beta_j)\right] \tag{3}$$

The log of the likelihood is a monotonic function of the likelihood and so the original ordering of the estimates between 2 and 3 is preserved. That is, the maximum likelihood estimates (MLE) of the log-likelihood are the same as the likelihood.

4 The Joint Maximum Likelihood Procedure

As denoted in Equation (1), there are two latent parameters: θ_i and β_j. All that is known is the response of person i to item j. If θ_i and the item responses were known, then we could simply maximize Equation (3) with respect to β_j. Conversely, if β_j were known and the item responses, but not θ_i, then we could simply maximize Equation (3) with respect to θ_i.

This suggests an iterative maximization process and is exactly how joint maximum likelihood (JML) works. The JML process proceeds in an iterative fashion by first estimating the ability parameters and then the item parameters. Operationally, the first step is to set all item parameters to 0 and then maximize Equation (3) with respect to θ. With these new ability estimates, we can now switch and maximize Equation (3) with respect to β_j. This process iterates between these steps until the difference in the estimates of the item parameters does not differ by more than .001 (default convergence criterion).

The process steps can be succintly described as:

1. Set $\beta_j = 0 \ \ \forall \ j$
2. Set $\frac{\ln \partial L}{\partial \theta} = 0$ and solve
3. Set $\frac{\ln \partial L}{\partial \beta} = 0$ and solve
4. Iterate between 2 and 3 until $abs|\beta_j^t - \beta_j^{t-1}| < .001 \ \ \forall \ j$

where the superscript denotes iteration t.

5 Estimation and Derivatives

Of course, the function is non-linear and this requires an iterative maximization process. The `jml` function uses Newton-Raphson steps and thus requires the first and second derivatives of the likelihood function. In the current implementation of `jml`, the analytic first and second derivatives are used in the Newton steps for the item parameters. However, the `optim` function is used to estimate ability parameters.

5.1 First and Second Derivatives of Item Parameters

For the item parameters, we find the first and second partial derivatives of the likelihood function with respect to β_j. The gradient vector is:

$$\mathbf{g} = \begin{bmatrix} -\sum_i (x_{i1} - P_{i1}) \\ -\sum_i (x_{i2} - P_{i2}) \\ \vdots \\ -\sum_i (x_{iN} - P_{iN}) \end{bmatrix} \tag{4}$$

where P_{ij} is the probability of a correct response by person i to item j as denoted in Equation (1).

The Hessian matrix for the items is a diagonal matrix of the following form:

$$\mathbf{H} = diag\left(-\sum_i P_{i1}[1-P_{i1}], -\sum_i P_{i2}[1-P_{i2}], \ldots, -\sum_i P_{iN}[1-P_{iN}]\right) \tag{5}$$

Using these derivatives, the Newton steps proceed as:

$$\mathbf{b}_{t+1} = \mathbf{b}_t - \mathbf{H}_t^{-1}\mathbf{g}_t \tag{6}$$

where $\mathbf{b}$ is the vector of estimated item parameters, $\mathbf{b} = (\hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_N)$, and the subscript denotes iteration t.

The code for the `jml` function currently builds the full $N \times N$ Hessian matrix and pre-multiplies the inverted Hessian by the gradient as represented algebraically in Equation (6). Because the matrix is diagonal, it is sufficient to divide each element of the gradient by its corresponding element in the Hessian. Hence, it is possible to use the vectorized calculations in `R` rather than the matrix algebra. However, experimenting with both showed that there was no computational efficiency in using the vectorized calculations rather than building the full Hessian as the number of item parameters tends to be very small. Hence, for transparency with the algebraic representation, I retain use of the matrix calculations.

The standard errors of the item parameters are simply derived from the diagonal elements of the Hessian matrix at convergence. That is, -1 * $\mathbf{H}_{jj}^{-.5}$ evaluated at $\mathbf{b}$ are the asymptotic standard errors.

5.2 First and Second Derivatives of Ability Parameters

It is also necessary to find the first and second partial derivatives of the ability parameters and use an iterative process in the maximization process. In contrast to the item parameters which uses the analytic derivatives, the function `theta.max` uses the `optim` function rather than analytic first and second derivatives. In an original implementation of the function `theta.max`, the analytic first and second derivatives were used within a `while` loop. However, the `R` code is sufficiently more compact using `optim`.

Nonetheless, for full transparency into how the JML process proceeds, the first and second derivatives of the ability estimates are presented. There is one difference in how items and ability parameters are estimated in `jml`. The process in the section above shows that the item parameters are estimated simultaneously, even though the estimate of b_j is independent of $b_{j'}$. But, for the ability parameters, the process proceeds one person at a time and not simultaneously as for the items.

Estimation proceeds in this manner for ability parameters as a convenience of the fact that the Hessian is diagonal. In the case of items, the Hessian tends to be small as the number of items is often small. But, the Hessian for persons would be $K \times K$, and its dimensions could be very large.

The first and second derivatives of the likelihood function with respect to θ are:

$$\frac{\partial L}{\partial \theta} = \sum_i (x_{ij} - P_{ij}) \tag{7}$$

$$\frac{\partial^2 L}{\partial^2 \theta} = \sum_i (1 - P_{ij})(-P_{ij}) \tag{8}$$

6 Centering and Correction for JML Bias

Consistent with Winsteps, the `jml` function removes the indeterminacy in the item parameters by centering the items on their mean. That is, the final item parameters are $\hat{\beta}_j = \hat{\beta}_j^* - \bar{\beta}$ where $\bar{\beta} = K^{-1} \sum_j \hat{\beta}_j$.

It is also well known that JML yields biased parameter estimates. The correction for bias proposed by Wright and Stone (1979) is implemented in the `jml` function as $\hat{\beta}_j \times K(K-1)$.

7 Item Fit Statistics

When the `jml` converges, it generates as output estimates of the item parameters, their standard errors, the sample size per item used in the estimation, and the Infit and Outfit statistics. These statistics are commonly used to evaluate the fit of the item parameters under the Rasch model.

In order to estimate both fit statistics it is first necessary to estimate a standardized variable, z, which is computed as:

$$z_{ij} = \frac{x_{ij} - P_{ij}}{\sqrt{\sigma_{ij}^2}} \tag{9}$$

where P_{ij} is the expected probability of a correct response for person i to item j from Equation (1) and σ_{ij}^2 is $P_{ij}(1 - P_{ij})$. From this, the fit statistics are estimated as:

$$Infit = \frac{\sum_i z_{ij}^2 \sigma_{ij}^2}{\sum_i \sigma_{ij}^2} \tag{10}$$

$$Outfit = N^{-1} \sum_i z_{ij}^2 \tag{11}$$

8 Estimating Examinee Ability

There are multiple methods for assessing examinee ability. **MiscPsycho** offers the user three common methods for estimating examinee ability via the `irt.ability` function: maximum likelihood estimation (MLE), *maximum a posteriori* (MAP), and the *expected a posteriori* (EAP).

Currently, many testing programs utilize a mixture of item formats including multiple choice items as well as constructed response items. Ability estimates are therefore based on the observed performance of examiness on these items. Hence, the likelihood function for a mixture of items can be expressed as:

$$L(\theta) = L(\theta)^{MC} L(\theta)^{CR} \tag{12}$$

where $L(\theta)^{MC}$ is the likelihood for dichotomously scored items:

$$L(\theta)^{MC} = \prod \left[c_i + \frac{1 - c_i}{1 + \exp[-Da_i(\theta - b_i)]} \right]^{x_i} \left[1 - c_i + \frac{1 - c_i}{1 + \exp[-Da_i(\theta - b_i)]} \right]^{1 - x_i} \tag{13}$$

where c_i is the lower asymptote of the trace function for the ith item (sometimes referred to as a guessing parameter), a_i is the slope of the trace function (i.e., the discrimination parameter), b_i is the location parameter, x_i is the binary response to the ith item (where 1 = correct), and D is a scaling factor commonly fixed at 1.7 to bring the logistic function into coincidence with the probit function.

$L(\theta)^{CR}$ is the likelihood for polytomously scored items based on the generalized partial credit model:

$$L(\theta)^{CR} = \prod \left[\frac{\exp \sum_{j=0}^{x} Da_i(\theta - \delta_{ij})}{\sum_{r=0}^{M} \left[\exp \sum_{j=0}^{r} Da_i(\theta - \delta_{ij}) \right]} \right] \tag{14}$$

where the notation is the same as above other than δ_{ij} which is the jth step for the ith item.

The function `irt.ability()` operationalizes these methods and provides the user with ability estimates assuming parameter estimates are known. The function is useful when there is a mixture of item formats (i.e., multiple choice and constructed response) or if there are only multiple choice or only constructed response. For instance, in cases where there is only multiple choice the likelihood function is simply:

$$L(\theta) = L(\theta)^{MC} \tag{15}$$

where $L(\theta)^{MC}$ is defined in Equation (13). This general expression of the likelihood offers the user flexibility as other common IRT models can be expressed as a special cases. For instance, the Rasch model is a special case of the 3PL when $a_i = 1 \; \forall \; i$, $c_i = 0 \; \forall \; i$, and $D = 1$. As such, this function can be used for many different IRT models when the appropriate constraints on the item parameters are imposed.

In maximum likelihood estimation, the goal is to maximize $L(\theta)$. This is available via the `irt.ability` function when `method = 'MLE'`. In some cases, there may exist prior information regarding the target population that can be used to update the current observed data. Hence, in the language of bayesian statistics, we may include a prior distribution, $g(\theta)$, which operationalizes this information:

$$MAP(\theta) = L(\theta)^{MC} L(\theta)^{CR} g(\theta) \tag{16}$$

When this prior is included and the function is maximized, the result is known as the MAP. This is available via the `irt.ability` function when `method = 'MAP'`. Within the `irt.ability` function it is always assumed that $g(\theta) \sim N(\mu, \sigma^2)$.

Rather than obtaining the MAP, one may prefer the mean of the posterior distribution, or the EAP. This is obtained as:

$$EAP(\theta) = \frac{\int_{-\infty}^{\infty} \theta L(\theta) g(\theta) d\theta}{\int_{-\infty}^{\infty} L(\theta) g(\theta) d\theta} \tag{17}$$

In Equation (17) there is no benefit of conjugacy, therefore the integral must be approximated. In the `irt.ability` function, this is approximated via Gauss-Hermite quadrature as:

$$EAP(\theta) \approx \frac{\sum_{i=1}^{Q} \theta_i L(\theta_i) w_i}{\sum_{i=1}^{Q} L(\theta_i) w_i} \tag{18}$$

where θ_i is node i (quadrature point) and w_i is the weight at node i. This is available in `irt.ability` when `method='EAP'`. The weights and nodes used in the computation are provided via the `gauss.quad.prob` function in the **statmod** package. The `irt.ability` function allows the user to change the number of quadrature points used in the approximation.

The user should keep in mind that EAP estimates are conceptually different than the MLE or the MAP. Both the MLE and the MAP are the result of an iterative maximization procedure whereas the EAP is the result of a non-iterative integral.

The user should be aware that, with the 3PL, maximization of the objective function may yield a local and not a global maximum. Hence, both the MAP and MLE are subject to this condition (only under the 3PL and GPCM as the 1- and 2PL are always unimodal). The user may implement different starting values via the `start_val` control option to determine whether a global maximum has been reached.

However, the EAP estimate is not subject to this condition and will always provide the posterior mean. Good approximations of the mean are dependent on the number of quadrature points used. The default is 149 as this has been found to provide excellent approximations in test cases, however the user should experiment to determine whether this proves true in different scenarios.

9 Plausible Values

Suppose we desire random samples from the following posterior density:

$$p(\theta|x, \hat{\boldsymbol{\eta}}) = \frac{L(\theta) g(\theta)}{\int L(\theta) g(\theta) d\theta} \tag{19}$$

where $\hat{\boldsymbol{\eta}}$ are estimates of the item response parameters and $\boldsymbol{x_i}$ is the vector of observed responses to all items for the ith individual, $L(\theta)$ is the likelihood as expressed in Equation (12) and $g(\theta) \sim N(\mu, \sigma^2)$. Given the lack of conjugacy between the data likelihood and the prior distribution, sampling from the posterior is difficult as its parametric form is unknown.

However, there are multiple methods that can be used to choose these samples. Methods used by the National Assessment of Educational Progress (NAEP) have assumed that $p(\theta|x, \hat{\boldsymbol{\eta}}) \sim N(\mu, \sigma^2)$ where μ and σ^2 are the EAP mean and variance. In this case, sampling is easy since the variates can be drawn from a normal distribution. However, this is

overly simplistic as it samples from a normal as an approximation given the complexity of the posterior.

Another option is to use an MCMC algorithm, such as Rejection Sampling, to sample from Equation (19). This is the method implemented in the `plaus.val` function. The algorithm as implemented proceeds as follows:

1. Draw a random variate, θ_i^*, from g(θ) $\sim$ N(0,1).
2. Draw a random variate, U_i, from $U_i \sim$ U[0,1].
3. Compute $r_i = p(\theta_i^*|x, \hat{\boldsymbol{\eta}})/$ [M * g(θ_i^*)]
4. If $U_i \leq r_i$ accept θ_i^* as a draw from $p(\theta|x, \hat{\boldsymbol{\eta}})$, else return to step 1

In this rejection sampling algorithm M is a constant that can be arbitrarily chosen so long as $M > 1$ and the normalizing constant in Equation (19) is computed using Gauss-Hermite quadrature as illustrated in Equation (18). Gelman et al (2004) suggest the constant M must have a known bound such that $p(\theta|x, \hat{\boldsymbol{\eta}})/g(\theta) \leq M \ \forall \ \theta$. The user can tune the acceptance rate by choosing different values for M via the `choose.M` function.

By default, the function returns five random draws from the posterior density, although more draws can be chosen. In fact, should the user choose many random draws, these variates can be used to form an empirical distribution of the posterior such that the mean of these variates is the EAP and the variance is $var(EAP)$.

10 Classification Accuracy: Integration over the Posterior

In educational testing situations, it is common to identify a point on the theta scale (ability scale) at which point a student must score in order to be considered "Proficient", which is denoted as γ. Hence, for scores below the cutpoint γ, we compute the probability that an individual with observed score $\theta_i < \gamma$ is truly proficient as:

$$p_i(\theta^* > \gamma|\theta < \gamma) = \frac{\int_\gamma^\infty L(\theta)g(\theta|\mu,\sigma)d\theta}{\int_{-\infty}^\infty L(\theta)g(\theta|\mu,\sigma)d\theta} \tag{20}$$

where θ^* is the unobserved true score, θ is the observed score on the proficiency scale, γ is the cut score required for passing, $L(\theta)$ is the data likelihood given the item parameters as expressed in Equation (12), and $g(\theta)$ is a normal population distribution. For individuals with observed scores at or above the proficient cut point we compute the probability that an individual at score $\theta_i \geq \gamma$ is truly not proficient as:

$$p_i(\theta^* < \gamma|\theta \geq \gamma) = \frac{\int_{-\infty}^\gamma L(\theta)g(\theta|\mu,\sigma)d\theta}{\int_{-\infty}^\infty L(\theta)g(\theta|\mu,\sigma)d\theta} \tag{21}$$

The integrals in Equation (20) and Equation (21) are evaluated using Gaussian quadrature. The nodes and weights used to evaluate these integrals are derived from the `gauss.quad`

and `gauss.quad.prob` functions in the **statmod** package. Currently, the functions used rely on 49 quadrature points. This was found to provide excellent approximations to the integrals when compared to the `Nintegrate` procedures in Mathematica.

For the numerator the following Gauss-Legendre quadrature is used if the student scored below the proficient cut point:

$$\begin{aligned} f(y_i) &= L(y_i + \gamma)g(y_i + \gamma|\mu, \sigma^2) && (22) \\ \int_{\gamma}^{\infty} L(\theta)g(\theta|\mu, \sigma)d\theta &\approx \sum_{i=1}^{Q} f(y_i)e^{y_i}w_i && (23) \end{aligned}$$

where γ is the proficient cut point, y_i is node $i = (1, \ldots, Q)$, and w_i is the weight at node i. Similarly, the following is used if the student scored above the proficient cut point:

$$\begin{aligned} f(y_i) &= L(\gamma - y_i)g(\gamma - y_i|\mu, \sigma^2) && (24) \\ \int_{-\infty}^{\gamma} L(\theta)g(\theta|\mu, \sigma)d\theta &\approx \sum_{i=1}^{Q} f(y_i)e^{y_i}w_i && (25) \end{aligned}$$

The normalizing constant is subsequently evaluated using Gauss-Hermite quadrature as:

$$\int_{-\infty}^{\infty} L(\theta)g(\theta|\mu, \sigma)d\theta \approx \sum_{i=1}^{Q} L(y_i)w_i \quad (26)$$

where w_i are weights derived from a Gaussian probability distribution with parameters μ, σ.

11 Examples

The following section illustrates sample use of the functions in the **MiscPsycho** package. As a first step, I use the `simRasch` function to generate sample data for 200 individuals to 10 test items. The default values of `mu` and `sigma` are 0 and 1 for the distribution of abilities.

```
> set.seed(1)
> dat <- simRasch(200, 10)
```

The `simRasch` function returns three values in a list:

```
> str(dat)
```

```
List of 3
 $ data              :'data.frame':        200 obs. of  10 variables:
  ..$ V1 : num [1:200] 0 0 0 0 0 0 0 1 0 0 ...
  ..$ V2 : num [1:200] 0 1 1 1 1 1 1 1 1 1 ...
  ..$ V3 : num [1:200] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ V4 : num [1:200] 0 0 0 0 0 0 0 0 1 0 ...
```

```
  ..$ V5 : num [1:200] 0 0 0 0 0 0 0 1 0 1 ...
  ..$ V6 : num [1:200] 1 0 1 1 0 0 0 0 0 0 ...
  ..$ V7 : num [1:200] 1 1 1 1 0 0 1 1 1 0 ...
  ..$ V8 : num [1:200] 0 0 0 1 0 0 0 0 1 0 ...
  ..$ V9 : num [1:200] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ V10: num [1:200] 0 0 0 0 0 0 0 0 0 0 ...
 $ generating.values: num [1:10]  0.953 -1.890  2.726  2.387  2.662 ...
 $ theta            : num [1:200] -0.626  0.184 -0.836  1.595  0.330 ...
```

These values are `data`, which are the item responses, `generating.values` which are the true values of the item parameters drawn from a $U(-3, 3)$ distribution and `theta` which are the true ability estimates drawn from a $N(\mu, \sigma)$ distribution.

11.1 Estimating Reliability

It is often useful to examine these data prior to running the IRT model. We can use the `alpha` and `alpha.Summary` functions to examine the reliability.

```
> alpha(dat$data)

[1] 0.4942997
```

The `alpha` function for the simulated data in this example returns a value of 0.49. Now, it may be useful to further examine what the reliability would be if item j were removed from the test using the `alpha.Summary` function.

```
> alpha.Summary(dat$data)

Below is what alpha *would be* if the item were removed

   Item     alpha
1     1 0.3988681
2     2 0.4895385
3     3 0.4856901
4     4 0.4467352
5     5 0.5122087
6     6 0.4429340
7     7 0.4554410
8     8 0.4463020
9     9 0.4954350
10   10 0.4805648
```

The output from `alpha.Summary` shows what the reliability of the test would be if an item were removed. For example, with these sample data, the reliability would be 0.4 if item 1 were removed.

11.2 Classical Item Analysis

Another preliminary way to examine the data prior to running the IRT model is to examine classical item statistics such as p-values and point-biserial correlations. The p-values are simply the means of the items over students. The point-biserial is the correlation of item j with the total score where the total score excludes item j. These statistics are accesible via the `classical` function as follows:

```
> classical(dat$data)
```

```
    p_values Point_Biserial
V1     0.300     0.37466067
V2     0.850     0.13301172
V3     0.105     0.13991013
V4     0.135     0.27627535
V5     0.085     0.02100712
V6     0.290     0.27187465
V7     0.680     0.24196345
V8     0.205     0.26622928
V9     0.935     0.08973595
V10    0.100     0.15985898
```

11.3 Estimating Rasch Parameters via JML

Now that we have examined our data, we can proceed with IRT estimation using the `jml` function. This function has three arguments: `dat`, `con`, and `bias`. `dat` is simply a data frame or matrix containing the item responses, `con` is the convergence criterion which is set at .001 as a default, and `bias` is a correction for bias argument. Its default value is `FALSE`.

Use of the function is simple and proceeds as follows with our simulated data:

```
> results <- jml(dat$data)
```

```
Convergence was reached in 10 iterations
       params        SE   N     Infit    Outfit
1   0.3065984 0.1797695 200 0.8445197 0.7426041
2  -3.2548110 0.2283092 200 1.1214078 0.8945367
3   1.9855983 0.2518549 200 1.0682641 1.0800688
4   1.6337026 0.2295986 200 0.9388617 0.7540313
5   2.2656415 0.2732766 200 1.2206078 1.7613372
6   0.3722937 0.1812322 200 0.9833109 0.8102370
7  -1.9282549 0.1792047 200 0.9405683 0.8383174
8   0.9890192 0.1996367 200 0.9775109 0.8117660
9  -4.4212190 0.3165415 200 1.0314255 1.1741563
10  2.0514313 0.2565722 200 1.0393149 1.0662122
```

The output shows the number of iterations required to reach convergence and gives a data frame containing the item parameters (`params`), the standard errors (`SE`), sample size (`N`), and the Infit and Outfit statistics.

11.4 Generating Score Conversion Tables

Now that the item parameters are estimated, it is possible to develop a score conversion table. A score conversion table gives the ability estimate for an individual with a total raw score of $X = \sum_j x_{ij}$. Since total score is a sufficient statistics for the Rasch model all individuals with the same raw score have the same ability estimate. An ability estimate is not generated for individuals with all items correct or zero items correct. That is because it is not possible to generate a maximum likelihood estimate (MLE) for these scores as the likelihood function is unbounded. The score conversion table is developed using the item parameters in the object `results` as:

```
> scoreCon(results$params)

  Raw.Score      Theta    SE
1         1 -4.1364471 1.315
2         2 -2.6352075 1.171
3         3 -1.3586950 1.077
4         4 -0.3470961 0.935
5         5  0.4312601 0.839
6         6  1.0963829 0.799
7         7  1.7336004 0.805
8         8  2.4266261 0.873
9         9  3.3621340 1.106
```

11.5 Estimating Examinee Ability

The prior examples work for the basic Rasch model since the total raw score is a sufficient statistic and there is a one-to-one relationship between the raw score total and the MLE. However, **MiscPsycho** also includes a function `irt.ability` that can be used to estimate the MLE, MAP, or EAP for any IRT model based on the 3-PL or Generalized Partial Credit Model (GPCM). In other words, the the model can include only dichotomous items, only polytomous items, or a mixture of item types.

In order to use this function, we must first organize the estimates of the item parameters into a list of lists. This task is simple, but a little prescriptive. For the first example, assume we have only two dichotomous test items based on the 2PL. Assume the item parameters for the first item are $a_1 = 1, b_1 = 0$ and for the second item $a_2 = 2, b_2 = 1$. Because this is a 2PL, the lower asymptote for both items is fixed at 0. Given these estimates, we can build the list as follows:

```
> alpha <- c(1, 2)
> beta <- c(0, 1)
> guess <- c(0, 0)
> params <- list("3pl" = list(a = alpha, b = beta, c = guess),
+     gpcm = NULL)
> params
```

```
$`3pl`
$`3pl`$a
[1] 1 2

$`3pl`$b
[1] 0 1

$`3pl`$c
[1] 0 0


$gpcm
NULL
```

In this example, we have no polytomous items, hence the list for the `GPCM` is `NULL`, denoting the list is empty. Once the list is created, use of the `irt.ability` function is simple. Assume we have an individual with a response pattern of `correct, incorrect`.

We can create a vector x with the observed responses to these items. In this example, there are only two items and they are both dichotomous. So, we use `ind.dichot = c(1,2)` which denotes that items 1 and 2 in the vector x are multiple choice.

```
> x <- c(1, 0)
> irt.ability(x, params, ind.dichot = c(1, 2), method = "MLE")

[1] 0.4923267

> irt.ability(x, params, ind.dichot = c(1, 2), method = "MAP")

[1] 0.3191912

> irt.ability(x, params, ind.dichot = c(1, 2), method = "EAP")

[1] 0.1919161
```

Note that simply changing the argument to `method` permits for us to estimate the MLE, MAP, or the EAP. The default values for the population parameters (i.e., priors) is $N(0, 1)$. However, changing these is also simple using the list of controls as follows:

```
> irt.ability(x, params, ind.dichot = c(1, 2), method = "MAP",
+     control = list(mu = 0.5, sigma = 1.2))

[1] 0.4942363
```

In the example above, the MAP is estimated using a $N(.5, 1.2)$ prior. The use of the argument `ind.dichot` is very simple. Assume we have a vector x such as `x <- c(0,3,1)` where item 1 is multiple choice, item 2 is polytomous, and item 3 is multiple choice. In this hypothetical case, the argument would be specified as `ind.dichot = c(1,3)` denoting that items 1 and 3 in the vector x are dichotomous.

Now, it may be the case that there is a mixture of items including dichotomous and polytomous. In this case, we organize the list of lists as follows:

```
> params <- list("3pl" = list(a = c(1, 1), b = c(0, 1),
+     c = c(0, 0)), gpcm = list(a = c(1, 1), d = list(item1 = c(0,
+     1, 2, 3, 4), item2 = c(0, 0.5, 1, 1.5))))
```

The only difference between this example and the first is that the list for `gpcm` is no longer `NULL`. It indeed contains two polytomous items. Note that $d_{1i} = 0$ (the first step category for item i is fixed at 0) for every item. Now, in this example, we create the vector x. Again, the first two items are dichotomous, but the last two are polytomous. Hence, the scores are `incorrect, correct, scored in category 2,scored in category 2`.

```
> x <- c(0, 1, 2, 2)
> irt.ability(x, params, ind.dichot = c(1, 2), method = "MLE")

[1] 0.8270681

> irt.ability(x, params, ind.dichot = c(1, 2), method = "MAP")

[1] 0.6762342

> irt.ability(x, params, ind.dichot = c(1, 2), method = "EAP")

[1] 0.6408158
```

The calls to `irt.ability` resembles the calls in the first example when there were only dichotomous items. That is because the dichotomous scores in the vector x are again in positions 1 and 2. Hence we again use `ind.dichot = c(1,2)`.

To complete our example, assume all items are polytomous. In this case, we organize the list as:

```
> params <- list("3pl" = NULL, gpcm = list(a = c(1, 1),
+     d = list(item1 = c(0, 1, 2, 3, 4), item2 = c(0, 0.5,
+         1, 1.5))))
> irt.ability(c(2, 3), params, method = "MLE")

[1] 1.401820

> irt.ability(c(2, 3), params, method = "MAP")

[1] 1.101404

> irt.ability(c(2, 3), params, method = "EAP")

[1] 1.091514
```

Note that the list of `'3pl'` is `NULL` and we do not use the argument `ind.dichot`. Given the way the likelihood function is expressed, the function `irt.ability` is extremely flexible and can be used to estimate ability using many different IRT models. For example, the 3PL reduces to the Rasch model for dichotomous items when $a_i = 1 \; \forall \; i$, $c_i = 0 \; \forall \; i$, and $D = 1$. As such, we can go back and use the b parameters estimated using `jml` in the prior example and use `irt.ability` as follows:

```
> params <- list("3pl" = list(a = rep(1, 10), b = results$params,
+     c = rep(0, 10)), gpcm = NULL)
> x <- c(1, 1, 1, 1, 1, 0, 0, 0, 0, 0)
> irt.ability(x, params, ind.dichot = c(1:10), method = "MLE",
+     control = list(D = 1))

[1] 0.4314441
```

If you go back to the score conversion table created using `scoreCon` in the prior example, you can see that the ability estimate associated with a raw score of 5 is `.43`. Hence, both `irt.ability`, `scoreCon`, and `theta.max` give exactly the same results. However, `irt.ability` is much more flexible than the other functions.

In fact, we can even constrain certain parameters for the GPCM such that it too reduces to the Rasch model and use `irt.ability` as follows:

```
> tt <- as.list(results$param)
> ll <- lapply(tt, function(x) c(0, x))
> params <- list("3pl" = NULL, gpcm = list(a = rep(1, 10),
+     d = ll))
> x <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
> irt.ability(x, params, method = "MLE", control = list(D = 1))

[1] 0.4314441
```

The reason this works is because the GPCM reduces to Master's Partial Credit Model when the $a = 1 \; \forall \; i$ and $D = 1$ and Master's Partial Credit Model reduces to the Rasch model when there are two categories. Note, for this to work under this parameterization, a "correct" score means the individual scored in category 2 and an incorrect response means the individual scored in category 1.

This example is provided simply to illustrate the flexibility of this function for estimating θ when certain constraints are placed on the item parameters. Of course, it would be unreasonable to estimate ability estmates for the Rasch model as this requires more work that necessary. However, it clearly illustrates how IRT models are connected and gives the user greater flexibility.

11.6 Sampling from the Posterior

`R` provides many built in functions for drawing random samples from a distribution. For example, `rnorm` or `runif` draw random variates from a normal or a uniform distribution. However, the Bayesian IRT model expressed in Equation (19) has no known form and sampling from it is difficult. **MiscPsycho** provides the `plaus.val` function that uses rejection sampling to draw random variates from the IRT posterior.

This function is also simple to use and its arguments are almost exactly the same as those used in `irt.ability`. for example, assume we have the following item parameters organized as a list of lists. We can use the `plaus.val` function to draw random draws from the posterior as follows:

```
> params <- list("3pl" = list(a = c(1, 1), b = c(0, 1),
+     c = c(0, 0)), gpcm = list(a = c(1, 1), d = list(item1 = c(0,
+     1, 2, 3, 4), item2 = c(0, 0.5, 1, 1.5))))
> plaus.val(x = c(0, 1, 2, 2), params = params, ind.dichot = c(1,
+     2))

[1] 0.9074444 0.3432524 0.7250432 1.3044971 0.4565303
```

Note, that by default the function returns five random draws as is done in NAEP. However, this can be modified via the `PV` argument as:

```
> aa <- plaus.val(x = c(0, 1, 2, 2), params = params, ind.dichot = c(1,
+     2), PV = 1000)
> mean(aa)

[1] 0.6642396
```

Now, we can compare the mean of these variates to the EAP estimate:

```
> irt.ability(x = c(0, 1, 2, 2), params = params, ind.dichot = c(1,
+     2), method = "EAP")

[1] 0.6408158
```

I do not proceed with an example here, but one could easily apply this function over many examiness, generate plausible values for each examinee, and subsequently study the population characteristics of the examinees using the means of the plausible values.

11.7 Posterior Density Function

Another useful function in **MiscPsycho** is `posterior`. Just as `dnorm` is the density for a normal distribution `posterior` is the density for the IRT posterior. Suppose we desire the density at $\theta = 1$. We can simply use this function as:

```
> posterior(x = c(0, 1, 2, 2), theta = 1, params = params,
+     ind.dichot = c(1, 2))

[1] 0.6755286
```

11.8 Classification Accuracy

One last function that may be useful is the `class.acc` function. This function can be used to perform integration over the posterior distribution to identify the proportion of the distribution that falls above (or below) or specific cut point on the ability scale.

Simply to illustrate use of the function, assume $\gamma = 0$.

```
> head(dat$data)
```

```
  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
1  0  0  0  0  0  1  1  0  1   0
2  0  1  0  0  0  0  1  0  1   0
3  0  1  0  0  0  1  1  0  1   0
4  0  1  0  0  0  1  1  1  1   0
5  0  1  0  0  0  0  0  0  1   0
6  0  1  0  0  0  0  0  0  1   0
```

In looking at the raw data, we can see that individual 1 has a raw score of 3, which corresponds to an ability estimate of -1.36 from the score conversion table. So, we can ask what is the probability that this individual's true score is above 0 (γ) using the `class.acc` function as follows:

```
> x <- as.numeric(dat$data[1, ])
> params <- list("3pl" = list(a = rep(1, 10), b = results$params,
+     c = rep(0, 10)), gpcm = NULL)
> rr <- class.acc(x, prof_cut = 0, params = params, ind.dichot = c(1:10),
+     aboveC = TRUE, control = list(D = 1))
> rr

[1] 0.1602521
```

So, we know that the probability is 16 percent that this individuals true score is above 0. Now, we can use the function in the other direction and also ask, what is the probability that an individual's true score is below 0. From the raw data we see that individual 4 has a raw score of 5 which corresponds to an ability estimate of .43. So, we can use the function as follows:

```
> x <- as.numeric(dat$data[4, ])
> rr <- class.acc(x, prof_cut = 0, params = params, ind.dichot = c(1:10),
+     aboveC = FALSE, control = list(D = 1))
> rr

[1] 0.3623349
```

This shows that there is a 36 percent probability that this individuals true score is below 0.

This function is general and will also work with other IRT models. Here we revisit an example using the 2PL.

```
> a <- c(1.45, 1.84, 2.55, 2.27, 3.68, 4.07, 2.26, 1.87,
+     2.19, 1.33)
> b <- c(-0.6, -0.82, -1.6, -0.87, -1.41, -1.33, -1.16,
+     -0.11, -0.64, -1.23)
> params <- list("3pl" = list(a = a, b = b, c = rep(0,
+     10)), gpcm = NULL)
> x <- c(rep(0, 9), 1)
> rr <- irt.ability(x, params, ind.dichot = c(1:10), method = "EAP")
> rr
```

```
[1] -1.851890
```

So, we see that the EAP for this individual is -1.85. We can use the `class.acc` function to ask what proportion of the posterior density falls above -1.5 on the theta scale:

```
> rr <- class.acc(x, prof_cut = -1.5, params, ind.dichot = c(1:10),
+     aboveC = TRUE)
> rr
```

```
[1] 0.09574338
```

This suggests there is only about a 10 percent probability that the true score for this individual is above $\theta = -1.5$.