MiscPsycho

An R Package for Miscellaneous Psychometric Analyses

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

November 18, 2009

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:

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

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 Classical Item Analysis

It is often of interest to examine item means (p-values), point-biserial correlations, and estimates of reliability. Functions for these estimates are all provided in this package.

Psychometricians often compare p-values to examine if there are differences between items. For instance, in a common item linking design one may compare the p-value for item i appearing on one test form with the p-value for item i appearing on a different test form. Sometmies, these comparisons take the form of a t-test, thus requiring estimates of the item standard errors.

Educational testing situations, however, tend to reflect sampling designs that are more complex than simple random samples. Consequently, students within clusters are more

similar to each other than they are to students in other clusters. This can be formally expressed as $cov(\epsilon_{j(i)}, \epsilon_{j(i')}) \neq 0 \; \forall \; i \neq i'$, where the notation $j(i)$ denotes student i in cluster j. Consequently, a person determining the standard errors of the item means must consider this cluster sample design to appropriately capture the true sampling variance of the items.

Under this cluster-sampling design, the appropriate procedure is to estimate design-consistent standard errors. This is accomplished by first defining the ratio estimator of the mean as:

$$f(Y) = \frac{Y}{N}, \tag{12}$$

where Y is the total of the variable and N is the population size. Treating both Y and N as random variables, a first-order Taylor series expansion of the ratio estimator $f(Y)$ can be used to derive the design-consistent variance estimator as:

$$var(f(Y)) = \left[\frac{\partial f(Y)}{\partial Y}, \frac{\partial f(Y)}{\partial N}\right] \left[\begin{array}{cc} var(Y) & cov(Y,N) \\ cov(Y,N) & var(N) \end{array}\right] \left[\frac{\partial f(Y)}{\partial Y}, \frac{\partial f(Y)}{\partial N}\right]^T \tag{13}$$

where

$$\frac{\partial f(Y)}{\partial Y} = \frac{1}{N}, \tag{14}$$

$$\frac{\partial f(Y)}{\partial N} = -\frac{Y}{N^2}, \tag{15}$$

$$var(Y) = \frac{k}{k-1} \sum_{j=1}^{k} (\hat{Y}_j - \hat{Y}_{..})^2, \tag{16}$$

$$\hat{Y}_j = \sum_{i=1}^{N_j} \hat{Y}_{j(i)}, \tag{17}$$

$$\hat{Y}_{..} = k^{-1} \sum_{j=1}^{k} \hat{Y}_j, \tag{18}$$

$$var(N) = \frac{k}{k-1} \sum_{j=1}^{k} (\hat{N}_j - \hat{N}_{..})^2, \tag{19}$$

$$\hat{N}_{..} = k^{-1} \sum_{j=1}^{k} \hat{N}_j, \tag{20}$$

$$cov(Y,N) = \frac{k}{k-1} \sum_{j=1}^{k} (\hat{Y}_j - \hat{Y}_{..})(\hat{N}_j - \hat{N}_{..}), \tag{21}$$

where j indexes cluster $(1, 2, \ldots, k)$, $j(i)$ indexes the ith member of cluster j, and N_j is the total number of members in cluster j.

Substituting these statistics into Equation (13) and expanding gives an approximate estimate of the variance of $f(Y)$ as:

$$var(f(Y)) = \frac{var(Y)N^2 + var(N)Y^2 - 2cov(Y,N)NY}{N^4} \tag{22}$$

The standard error is then taken as:

$$se = \sqrt{var(f(Y))} \tag{23}$$

The **survey** package in `R` has many useful functions for survey data. However, design-consistent standard errors for item means can also be easily obtained through the use of the `classical` function in this package.

9 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:

$$L(\theta) = L(\theta)^{MC} L(\theta)^{CR} \tag{24}$$

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

$$L(\theta)^{MC} = \prod \left[c_i + \frac{1 - c_i}{1 + \exp[-Da_i(\theta - b_i)]} \right]^{x_i} \left[1 - c_i + \frac{1 - c_i}{1 + \exp[-Da_i(\theta - b_i)]} \right]^{1 - x_i} \tag{25}$$

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:

$$L(\theta)^{CR} = \prod \left[\frac{\exp \sum_{j=0}^{x} Da_i(\theta - \delta_{ij})}{\sum_{r=0}^{M} \left[\exp \sum_{j=0}^{r} Da_i(\theta - \delta_{ij}) \right]} \right] \tag{26}$$

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:

$$L(\theta) = L(\theta)^{MC} \tag{27}$$

where $L(\theta)^{MC}$ is defined in Equation (25). 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:

$$MAP(\theta) = L(\theta)^{MC} L(\theta)^{CR} g(\theta) \tag{28}$$

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:

$$EAP(\theta) = \frac{\int_{-\infty}^{\infty} \theta L(\theta) g(\theta) d\theta}{\int_{-\infty}^{\infty} L(\theta) g(\theta) d\theta} \tag{29}$$

In Equation (29) 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:

$$EAP(\theta) \approx \frac{\sum_{i=1}^{Q} \theta_i L(\theta_i) w_i}{\sum_{i=1}^{Q} L(\theta_i) w_i} \tag{30}$$

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 standard error of $EAP(\theta)$ is taken as the standard deviation of the posterior distribution in Equation (29). Formally, the standard deviation of the posterior is:

$$SD[EAP(\theta)] = \left[\frac{\int_{-\infty}^{\infty} L(\theta) g(\theta) (\theta - EAP(\theta))^2 d\theta}{\int_{-\infty}^{\infty} L(\theta) g(\theta) d\theta} \right]^{1/2} \tag{31}$$

Again, the integrals are approximated using Gauss-Hermite quadrature and so the implementation of Equation (31) is based on the following:

$$SD[EAP(\theta)] \approx \left[\frac{\sum_{i=1}^{Q} (\theta_i - EAP(\theta))^2 L(\theta_i) w_i}{\sum_{i=1}^{Q} L(\theta_i) w_i} \right]^{1/2} \tag{32}$$

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.

10 Plausible Values

Suppose we desire random samples from the following posterior density:

$$p(\theta|x, \hat{\boldsymbol{\eta}}) = \frac{L(\theta)g(\theta)}{\int L(\theta)g(\theta)d\theta} \tag{33}$$

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 (24) 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 (33). 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 (33) is computed using Gauss-Hermite quadrature as illustrated in Equation (30). 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 $var(EAP)$.

11 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:

$$p_i(\theta^* > \gamma | \theta < \gamma) = \frac{\int_{\gamma}^{\infty} L(\theta) g(\theta|\mu, \sigma) d\theta}{\int_{-\infty}^{\infty} L(\theta) g(\theta|\mu, \sigma) d\theta} \tag{34}$$

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 (24), 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:

$$p_i(\theta^* < \gamma | \theta \geq \gamma) = \frac{\int_{-\infty}^{\gamma} L(\theta) g(\theta|\mu, \sigma) d\theta}{\int_{-\infty}^{\infty} L(\theta) g(\theta|\mu, \sigma) d\theta} \tag{35}$$

The integrals in Equation (34) and Equation (35) 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:

$$f(y_i) = L(y_i + \gamma) g(y_i + \gamma | \mu, \sigma^2) \tag{36}$$

$$\int_{\gamma}^{\infty} L(\theta) g(\theta|\mu, \sigma) d\theta \approx \sum_{i=1}^{Q} f(y_i) e^{y_i} w_i \tag{37}$$

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:

$$f(y_i) = L(\gamma - y_i)g(\gamma - y_i|\mu, \sigma^2) \quad (38)$$

$$\int_{-\infty}^{\gamma} L(\theta)g(\theta|\mu, \sigma)d\theta \approx \sum_{i=1}^{Q} f(y_i)e^{y_i}w_i \quad (39)$$

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

$$\int_{-\infty}^{\infty} L(\theta)g(\theta|\mu, \sigma)d\theta \approx \sum_{i=1}^{Q} L(y_i)w_i \quad (40)$$

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

12 Excessive Similarity in Student Response Patterns

With multiple choice tests it is often of interest to examine how student patterns of responses compare to other students within a group to determine if answer copying may have occured. Many "cheating" detection methods have been proposed and the implementation here is based on the work of Wesolowsky (2000).

The goal of the method is relatively straightforward: identify pairs of students with response patterns too similar to have occured from random chance alone. The interested reader should consult the full text of the original article for complete details. However, I note here only a few computational highlights.

The problem first requires that we estimate the probability that student j will answer test item i correctly. This is estimated as:

$$\hat{p}_{ij} = (1 - (1 - r_i)^{a_j})^{a_j} \quad (41)$$

where r_i is the proportion of individuals within the class that answered the item correctly. The parameter a_j can be found as:

$$c_j = \frac{\sum_{i=1}^{q} \hat{p}_{ij}}{q} \quad (42)$$

where q is the number of items on the test and c_j is the proportion of questions answered correctly by student j. Once an estimate of a_j is available, we can subsequently estimate the probability of answering each item correctly because all other values are observed. Hence, the challenge is to obtain an estimate of a_j.

In the `cheat` function provided in this package, the user can choose from three different root finding algorithms: newton-raphson, bisection, or use Rs internal uniroot function.

13 Similar Student Index

The nearest neighbor functions in this package (called SSI for similar student index) were designed to estimate growth eprcentiles for individuals students conditional on where they

started. This is often needed because growth rates tend to vary for students conditional on their prior test scores. In other words, students who start in higher locations on the scale appear to grow at slower rates than students who start at lower locations on the scale.

Even though conditional norms for growth rates was the primary rationale for the development of this function, it is clearly general and can be used to construct conditional norms for any instance in which they are desired. The example below uses gain scores, but any score needing a conditional norm could be use din the same manner as the example below.

As mentioned, the motivation to use a nearest neighbor approach stems from the fact that growth rates for students tend to vary conditional on where the student began. Consequently, variability in the growth rates may be a function of certain psychometric properties of the scale and not a function of instructional efficacy. As such, it seems unwise to compare the growth rates for students who began in very different points on the score scale. Therefore, the goal is to compare student growth rates only to other students who started in the same location on the score scale.

Some approaches accomplish this task through the use of regression-based methods using the postest score as the outcome variable and condition on prior achievement scores. These approaches are known to introduce bias into the estimates of the fixed effects when analysts condition on variables measured with error. This effect is well-known, but conditioning on variables measured with error perists as a measurement practice nonetheless.

Here I propose an entirely different approach. While the method is different, I share the same motivation as others to compare growth rates among similar students. Given a student i and a set of j characteristics associated with student i, identify K other students that are most similar to student i on the set of j characteristics.

Given a suitable distance metric, the task is straight forward. Compute the distance between student i and all other students in the data, sort those distances, choose the K students in the data whose distance is closest to that of student i.

More formally stated, given a data set, M, with n data points in d-dimensional metric space, X, the aim is to identify the K data points nearest to a *query point* $q \in X$. Implementing this process requires a metric to measure the distance between the points, for which we use the following:

$$D_i = \sqrt{\sum_{j=1}^{K}(q_{ji} - q_{ji'})^2} \quad \forall \quad i \neq i', \quad j = 1, \ldots, K \tag{43}$$

The distance metric is based on the well-known Euclidean distance metric which yields the ℓ^2 norm when more than two variables are used. This algorithm is simple to implement, but can be computationally complex depending on the the size of n, the total of records in the database. Because the process iterates over each student in the data to find the nearest neighbor, the behavior of the algorithm expands with n and executes in $\mathcal{O}(n^2 \log n)$.

13.1 Computing Norms From K Nearest Neighbors

Equation (43) is used to identify a group of K similar students. Subsequently, this group is used as the norming group by which the student's percentile rank is determined. The percentile ranks are easily determined as:

$$z_i = \frac{\hat{g}_i - \bar{g}_m}{\sigma(g_m)} \tag{44}$$

where $\hat{g}_i$ is a student-level gain score, $\bar{g}_m$ is the mean of $\hat{g}_i$ for the N students selected to be in the mth similar student group, and $\sigma(g_m)$ is the standard deviation of $\bar{g}_m$. Because the procedure identifies a group of N students that varies for each comparison, we note that both $\bar{g}_m$ and $\sigma(g_m)$ vary when computing the z-score for every student as well.

Essentially, all that is needed to implement this procedure is a gain score statistic, g_i, and some set of variables that are used to identify other similar students. Those variables may be other test scores, demographic characteristics, or geographic characteristics of the students. Because our purpose here is to demonstrate the method for generating norms, we implement a straight forward method for estimating a student's gain. Virtually any other method for estimating g_i can be used for this process.

The statistic $\hat{g}_i$ is the centered gain score for student i computed as:

$$g_i = (\hat{\theta}_{gti} - \bar{\theta}_{gt}) - (\hat{\theta}_{gt-1,i} - \bar{\theta}_{gt-1}) \tag{45}$$

where $\hat{\theta}_{gti}$ is the observed score at time t for student i and $\bar{\theta}_{gt}$ is the state mean for grade g at time t. The scores are centered before computing gains because the vertical scale is not truly an interval scale across grades. For example, a gain of 20 scaled score points from grades 4 to 5 is not necessarily comparable to a gain of 20 points from grades 6 to 7. This centering is an attempt to ameleriote some of the problems associated with comparing gains across grades when the same scaling properties do not hold.

Percentile ranks for every z_i can be easily obtained from the quantiles of the normal distribution. Percentiles have the attractive property in that they are readily understood by users of the data, but they are not useful for arithmetic manipulation, such as calculating means. Z-scores, on the other hand, lack transparency to the user, but are useful for mathematical manipulation. Since percentiles can be readily obtained from the standard scores, it is useful to have both as we further demonstrate.

14 Equating with the Stocking Lord Procedure

Stocking-Lord is an iterative procedure that minimizes the distance between two test characteristic curves. This method has been implemented in via the function `SL` in this package as described in this section.

Treating θ as a continuous variable, it is possible to introduce a population distribution $f(\theta)$, integrate θ out of the function, and then perform the minimization as follows:

$$SL = \int \left[\sum_{i=1}^{I} p(\theta; a_{ia}, b_{ia}, c_{ia}) - \sum_{i=1}^{I} p\left(\theta; \frac{a_{ib}}{A}, Ab_{ib} + B, c_{ib}\right) \right]^2 f(\theta) d\theta \tag{46}$$

where i indexes item, $i = (1, \ldots, I)$, a and b index test forms, $f(\theta) \sim N(\mu, \sigma^2)$ is the mean and variance of the population distribution, and A and B are the linking constants.

The integral in the function SL cannot be evaluated easily. For that reason it is approximated using Gauss-Hermite quadrature as follows:

$$SL \approx \sum_{q=1}^{Q} \left[\sum_{i=1}^{I} p(\theta_q; a_{ia}, b_{ia}, c_{ia}) - \sum_{i=1}^{I} p\left(\theta_q; \frac{a_{ib}}{A}, Ab_{ib} + B, c_{ib}\right) \right]^2 w_q \tag{47}$$

where q indexes quadrature point, $q = (1, \ldots, Q)$ and w_q is the weight at quadrature point q.

Minimization of the function SL with respect to A and B is implemented using Newton-Raphson iterations as:

$$\boldsymbol{L_{t+1}} = \boldsymbol{L_t} - \boldsymbol{H_t}^{-1}\boldsymbol{g_t}^T \tag{48}$$

where the subscript denotes iteration t and:

$$\boldsymbol{L} = [\hat{A}, \hat{B}] \tag{49}$$

$$\boldsymbol{g} = \left[\frac{\partial SL}{\partial A}, \frac{\partial SL}{\partial B}\right] \tag{50}$$

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 SL}{\partial A} & \frac{\partial^2 SL}{\partial B, \partial A} \\ \frac{\partial^2 SL}{\partial A, \partial B} & \frac{\partial^2 SL}{\partial B} \end{bmatrix} \tag{51}$$

The required derivatives for the 3-parameter logistic model are:

$$\begin{aligned} \frac{\partial SL}{\partial A} &= 2\left[\sum_{i=1}^{I} c_{ia} + \frac{1 - c_{ia}}{1 + \exp[-Da_{ia}(-b_{ia} - \theta)]} - \sum_{i=1}^{I} c_{ib} + \frac{1 - c_{ib}}{1 + \exp\left[\frac{-Da_{ib}(-B - Ab_{ib} + \theta)}{A}\right]}\right] \\ &\quad \times \sum_{i=1}^{I}\left[\frac{(1 - c_{ib})\exp\left[\frac{-Da_{ib}(-B - Ab_{ib} + \theta)}{A}\right]\left[\frac{Da_{bi}b_{bi}}{A} + \frac{Da_{ib}(-B - Ab_{ib} + \theta)}{A^2}\right]}{\left(1 + \exp\left[\frac{-Da_{ib}(-B - Ab_{ib} + \theta)}{A}\right]\right)^2}\right] \end{aligned}$$

$$\begin{aligned} \frac{\partial SL}{\partial B} &= \left[\sum_{i=1}^{I} c_{ia} + \frac{1 - c_{ia}}{1 + \exp[-Da_{ia}(-b_{ia} - \theta)]} - \sum_{i=1}^{I} c_{ib} + \frac{1 - c_{ib}}{1 + \exp\left[\frac{-Da_{ib}(-B - Ab_{ib} + \theta)}{A}\right]}\right] \\ &\quad \times 2\sum_{i=1}^{I}\left[\frac{Da_{bi}(1 - c_{bi})\exp\left[\frac{-Da_{ib}(-B - Ab_{ib} + \theta)}{A}\right]}{A\left(1 + \exp\left[\frac{-Da_{ib}(-B - Ab_{ib} + \theta)}{A}\right]\right)^2}\right] \end{aligned}$$

$$
\begin{aligned}
\frac{\partial^2 SL}{\partial A} &= \sum_{i=1}^{I}\left[\frac{(1-c_{ib})\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\left[\frac{Da_{bi}b_{bi}}{A}+\frac{Da_{ib}(-B-Ab_{ib}+\theta)}{A^2}\right]}{\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^2}\right]^2 \\
&\quad +2\left[\sum_{i=1}^{I}c_{ia}+\frac{1-c_{ia}}{1+\exp[-Da_{ia}(-b_{ia}-\theta)]}-\sum_{i=1}^{I}c_{ib}+\frac{1-c_{ib}}{1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]}\right] \\
&\quad \times\sum_{i=1}^{I}\left[\frac{(1-c_{ib})\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\left[\frac{-2Da_{bi}b_{bi}}{A^2}-\frac{2Da_{ib}(-B-Ab_{ib}+\theta)}{A^3}\right]}{\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^3}\right] \\
&\quad +\sum_{i=1}^{I}\left[\frac{(1-c_{ib})\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\left[\frac{Da_{bi}b_{bi}}{A}+\frac{Da_{ib}(-B-Ab_{ib}+\theta)}{A^2}\right]^2}{\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^3}\right] \\
&\quad -\sum_{i=1}^{I}\left[\frac{2(1-c_{ib})\exp\left[\frac{-2Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\left[\frac{Da_{bi}b_{bi}}{A}+\frac{Da_{ib}(-B-Ab_{ib}+\theta)}{A^2}\right]^2}{\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^3}\right]
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 SL}{\partial B} &= 2\sum_{i=1}^{I}\left[\frac{Da_{bi}(1-c_{bi})\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]}{A\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]^2\right)}\right]^2 \\
&\quad +2\left(\sum_{i=1}^{I}\left[\frac{D^2a_{bi}^2(1-c_{bi})\exp\left[\frac{-2Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]}{A^2\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^2}\right]\right. \\
&\quad \left.+\sum_{i=1}^{I}\left[\frac{2D^2a_{bi}^2(1-c_{bi})\exp\left[\frac{-2Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]}{A^2\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^3}\right]\right) \\
&\quad \times\left[\sum_{i=1}^{I}c_{ia}+\frac{1-c_{ia}}{1+\exp[-Da_{ia}(-b_{ia}-\theta)]}-\sum_{i=1}^{I}c_{ib}+\frac{1-c_{ib}}{1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]}\right]
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 SL}{\partial A, \partial B} &= 2\sum_{i=1}^{I}\left[\frac{Da_{bi}(1-c_{bi})\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]}{A\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^2}\right] \\
&\times\sum_{i=1}^{I}\left[\frac{(1-c_{ib})\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\left[\frac{Da_{bi}b_{bi}}{A}+\frac{Da_{ib}(-B-Ab_{ib}+\theta)}{A^2}\right]}{\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^2}\right] \\
&+2\left[\sum_{i=1}^{I}c_{ia}+\frac{1-c_{ia}}{1+\exp[-Da_{ia}(-b_{ia}-\theta)]}-\sum_{i=1}^{I}c_{ib}+\frac{1-c_{ib}}{1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]}\right] \\
&\times\sum_{i=1}^{I}\left[\frac{Da_{bi}(1-c_{ib})\exp\left[\left(\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right)\right]\left[\frac{Da_{bi}b_{bi}}{A}+\frac{Da_{ib}(-B-Ab_{ib}+\theta)}{A^2}\right]}{A\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^2}\right] \\
&-\sum_{i=1}^{I}\left[\frac{Da_{bi}(1-c_{bi})\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]}{A^2\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^2}\right] \\
&+\sum_{i=1}^{I}\left[\frac{2Da_{bi}(1-c_{ib})\exp\left[\frac{-2Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\left[\frac{Da_{bi}b_{bi}}{A}+\frac{Da_{ib}(-B-Ab_{ib}+\theta)}{A^2}\right]}{A\left(1+\exp\left[\frac{-Da_{ib}(-B-Ab_{ib}+\theta)}{A}\right]\right)^3}\right]
\end{aligned}
$$

Initial starting values for the linking constants A and B are taken from the mean/sigma transformation

$$A = \frac{\sigma(\hat{b}_b)}{\sigma(\hat{b}_a)} \tag{52}$$

$$B = \mu(\hat{b}_b) - A * \mu(\hat{b}_a) \tag{53}$$

15 Levenshtein Distance

R has many functions for pattern matching, especially those based on `grep`. However, this function is added to this package for a specific psychometric application–the merging of student databases to create longitudinal data files. The complete details demonstrating how to apply this function for merging student records to create longitudinal data files is found in Doran (in press) [1]

Assume we have two character strings that we wish to compare. The first string is `Bill Clinton` and the second string is `William Clinton`. The purpose of the LD procedure is to empirically determine how similar these two character strings are. In this example,

[1] Application of the Levenshtein Distance Metric For the Construction of Longitudinal Data Files. Educational Measurement: Issues and Practice.

the last name is exactly the same and no edits, insertions, or substitutions are necessary. The first name differs, however. Transforming `Bill` to `William` would require the following operations:

Operation 1: Substitute `W` for the `B`

Operation 2: Insert `i` after the `Will`

Operation 3: Insert `a` after the `Willi`

Operation 4: Insert `m` after the `Willia`

Four operations are needed to transform `Bill` to `William`; hence, the edit distance is 4. The edit distance is useful, but normalizing the distance to fall within the interval [0,1] is preferred because it is somewhat difficult to judge whether an LD of 4 suggests a high or low degree of similarity. For instance, a comparison of `Tim` to `Jane` yields an LD of 4 as does the `Tim` to `Timothy` comparison. However, the former is clearly a different comparison, whereas the latter is not. Our method for normalizing the LD is sensitive to this scenario and would return results that would indicate the latter is more similar than the former.

In our implementation, the Levenshtein distance is transformed to fall in this interval as follows:

$$LND = 1 - \frac{LD}{max(s1, s2)} \tag{54}$$

where LD is the edit distance and $max(s1, s2)$ denotes that we divide by the length of the larger of the two character strings. This normalization, referred to as the Levenshtein normalized distance (LND), yields a statistic where 1 indicates perfect agreement between the two strings, and a 0 denotes imperfect agreement. The closer a value is to 1, the more certain we can be that the character strings are the same; the closer to 0, the less certain. This normalized statistic is similar to the Jaro-Winkler method.

In addition to applying the LND, it is useful to determine the chances of observing an LND value of x or larger in a population of names. In other words, the desired inference is the probability that a comparison of two random character strings would yield an LND statistic of x. This probability can be used to help determine whether the LND statistic obtained between the two character strings is likely to occur from random chance.

For example, if a comparison of two strings returned an LND of .7 and the $\mathbb{P}(\text{LND} \geq .7) = .001$, we could be relatively confident that any two comparisons yielding an LND of .7 are similar names given that it is so unlikely that an LND = .7 would occur from a chance comparison of two random strings. On the other hand, if a comparison of two strings returned an LND of .3 and the $\mathbb{P}(\text{LND} \geq .3) = .6$, then we have some assurance that an LND of .3 yields an incorrect comparison given that an LND of .3 would occur about 60% of the time when two random strings are compared.

Given a large data set with many names (such as a student test score database), the following procedure can be used to obtain these probabilities empirically:

1. Take a random sample without replacement of n names from the data.

2. Compare each n name to all N student names in the data to obtain the LND.

3. Count the number of times the LND of x_i is observed.

4. Divide x_i by the total number of comparisons made to obtain $p(x_i)$.

Because the intended inference is $\mathbb{P}(x \geq x_i)$, we compute the cumulative probabilities as:

$$\mathbb{P}(x \geq x_i) = 1 - \sum_{x_i \leq x} p(x_i) \tag{55}$$

16 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 -1.89 2.726 2.387 2.662 ...
 $ theta            : num [1:200] -0.626 0.184 -0.836 1.595 0.33 ...
```

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.

16.1 Estimating Reliability

It is often useful to examine these data prior to running the IRT model. We can use the `alpha` function to examine the reliability of the test. Note, the `alpha.Summary` function has been deprecated. The `alpha` function now returns the info that was previously returned by `alpha.Summary`.

R users are often familiar with the formula methods used to specify a fitted model, such as the use of a formula in the `lm` function. Almost all functions in this package also make use of the formula methods as I demonstrate below.

```
> itemDat <- dat$data
> ff <- as.formula(paste("~", paste(names(itemDat), collapse = "+")))
> (aa <- alpha(ff, itemDat))

 Call:
alpha.formula(formula = ff, data = itemDat)

Cronbach's Coefficient Alpha: 0.4942997

> summary(aa)

Cronbach's Coefficient Alpha:

      alpha
[1,] 0.4943

Number of test items: 10

Conditional Alpha:

The data below show 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 `alpha` function for the simulated data in this example returns a value of 0.49 for the total test. The summary methods prints the conditional alpha, or the alpha that would be obtained if the item were removed from the test.

16.2 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.

The classical function also returns standard errors of the p-values. Our use of the function is simple:

```
> (aa <- classical(ff, data = itemDat))

Call:
classical.formula(formula = ff, data = itemDat)

Classical Item Analysis:
P-Values: 0.300 0.850 0.105 0.135 0.085
Point Biserial Correlations: 0.37466067 0.13301172 0.13991013 0.27627535 0.02100712

> summary(aa)

Call:
classical.formula(formula = ff, data = itemDat)

    P.Values Std.Errors PointBiserials
V1  0.300000   0.032404         0.3747
V2  0.850000   0.025249         0.1330
V3  0.105000   0.021677         0.1399
V4  0.135000   0.024164         0.2763
V5  0.085000   0.019720         0.0210
V6  0.290000   0.032086         0.2719
V7  0.680000   0.032985         0.2420
V8  0.205000   0.028546         0.2662
V9  0.935000   0.017432         0.0897
V10 0.100000   0.021213         0.1599
```

However, assume we have a grouping variable, such as a school or classroom and we wish to consider the correlation of students within clusters to obtain standard errors that reflect the sampling design. Our use of the function would simply add two arguments: `designSE` and `group`. The `designSE` simply tells our function to return design consistent standard errors. The `group` argument requires that we specify a variable in the data frame to serve as the grouping variable. In our example below, this is a variable called "group".

```
> itemDat$group <- gl(10, 20)
> (aa <- classical(ff, data = itemDat, design = TRUE, group = group))
```

```
Call:
classical.formula(formula = ff, data = itemDat, designSE = TRUE,     group = group)

Classical Item Analysis:
P-Values: 0.300 0.850 0.105 0.135 0.085
Point Biserial Correlations: 0.37466067 0.13301172 0.13991013 0.27627535 0.02100712
```

```
> summary(aa)
```

```
Call:
classical.formula(formula = ff, data = itemDat, designSE = TRUE,
    group = group)

    P.Values Std.Errors PointBiserials
V1  0.300000   0.025820         0.3747
V2  0.850000   0.022361         0.1330
V3  0.105000   0.032872         0.1399
V4  0.135000   0.030777         0.2763
V5  0.085000   0.015000         0.0210
V6  0.290000   0.028674         0.2719
V7  0.680000   0.035119         0.2420
V8  0.205000   0.029297         0.2662
V9  0.935000   0.013017         0.0897
V10 0.100000   0.014907         0.1599
```

16.3 Estimating Rasch Parameters via JML

Now that we have examined our data, we can proceed with IRT estimation using the `jml` function. This function follows the same convention as above now using the formula call to estimate model parameters. Estimating item parameters involves only a very simple call to `jml`:

```
> (fm1 <- jml(ff, itemDat, bias = TRUE))
```

```
Call:
jml.formula(formula = ff, data = itemDat, bias = TRUE)

Coefficients:
       [,1]
 [1,]   0.31
 [2,]  -3.25
 [3,]   1.99
 [4,]   1.63
 [5,]   2.27
 [6,]   0.37
 [7,]  -1.93
```

```
 [8,]   0.99
 [9,]  -4.42
[10,]   2.05
```

```
> summary(fm1)
```

```
Call:
jml.formula(formula = ff, data = itemDat, bias = TRUE)

Number of iterations to completion: 10
Number of individuals used in estimation: 200

      Estimate StdError    Infit Outfit
 [1,]  0.30660  0.17977  0.84452 0.7426
 [2,] -3.25481  0.22831  1.12141 0.8945
 [3,]  1.98560  0.25185  1.06826 1.0801
 [4,]  1.63370  0.22960  0.93886 0.7540
 [5,]  2.26564  0.27328  1.22061 1.7613
 [6,]  0.37229  0.18123  0.98331 0.8102
 [7,] -1.92825  0.17920  0.94057 0.8383
 [8,]  0.98902  0.19964  0.97751 0.8118
 [9,] -4.42122  0.31654  1.03143 1.1742
[10,]  2.05143  0.25657  1.03931 1.0662
```

```
> coef(fm1)
```

```
            [,1]
 [1,]  0.3065984
 [2,] -3.2548110
 [3,]  1.9855983
 [4,]  1.6337026
 [5,]  2.2656415
 [6,]  0.3722937
 [7,] -1.9282549
 [8,]  0.9890192
 [9,] -4.4212190
[10,]  2.0514313
```

The summary output provides statistic most users of Rasch models are familiar with including `Estimate`, which are the b-values of the items, `Std.Error` the standard errors of the b-values, and the `Infit`and `Outfit` statistics.

It may also be useful to examine some diagnostic plots of the items. Fitting an IRT model using the `jml` function returns an object of class "jml" and there is now a plot method associated with that class. Hence, users can explore the items via the following calls:

```
> plot(fm1)
> plot(fm1, all = FALSE, item = 3, pch = 2, lty = 1)
```

The first call cycles through each of the test items and prompts the user for input before proceeding to the next item. However, if the user wants to view one, and only one specific item, then we set `all = FALSE` and specify which item to view via the argument `item = 3`. In this case, we view item 3 in our data. The `plot.jml` function inherits other arguments from `par`, hence we can use arguments such as `pch` or `lty`.

16.4 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 can be easily created because the item parameters from the jml object can be easily accessed via the traditional `coef` extractor function used on fitted model objects. For instance,

```
> (ss <- scoreCon(coef(fm1)))

Coefficients:
[1]  -4.14  -2.64  -1.36  -0.35   0.43   1.10   1.73   2.43   3.36

> summary(ss)

      Estimate StdError Raw.Score
 [1,] -4.13645  1.31500         1
 [2,] -2.63521  1.17100         2
 [3,] -1.35869  1.07700         3
 [4,] -0.34710  0.93500         4
 [5,]  0.43126  0.83900         5
 [6,]  1.09638  0.79900         6
 [7,]  1.73360  0.80500         7
 [8,]  2.42663  0.87300         8
 [9,]  3.36213  1.10600         9
```

16.5 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 = coef(fm1),
+     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(coef(fm1))
> 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.

16.6 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 (33) 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.

16.7 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
```

16.8 Classification Accuracy

The `class.acc` 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 = coef(fm1),
+     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 $\theta = -1.5$.

16.9 Similar Student Index: Generating Conditional Norms

The `SSI` function implements the methods previously described for the similar student index. To demonstrate, assume we have three test scores for all students: a math score, a reading score, and a science score. Assume we desire to generate a math score norm for each student based on other students in the data that most resemble them. We first need a data file, such as the one below.

```
> tmp <- data.frame(ID = 1:100, mathScore = rnorm(100),
+     readScore = rnorm(100), scienceScore = rnorm(100))
```

Now to proceed, we require that we somehow operationalize the term "similar". That is, how do we decide who is most like the student in question? In this implementation, this is accomplished empirically using what I refer to as "conditioning" variables. In this case, suppose we want to construct a math norm for a student based on the 20 other students in the data whose reading and math scores most resemble those of our student (note, norms based on 20 other students are poorly estimated, this value of k is used only for the example).

Using our `SSI` function this is accomplished as:

```
> (result <- SSI(mathScore ~ readScore + scienceScore,
+     tmp, k = 20, id = ID, na.action = na.omit))

Iteration 1          1 percent complete
Iteration 2          2 percent complete
Iteration 3          3 percent complete
Iteration 4          4 percent complete
Iteration 5          5 percent complete
Iteration 6          6 percent complete
Iteration 7          7 percent complete
Iteration 8          8 percent complete
Iteration 9          9 percent complete
Iteration 10          10 percent complete
Iteration 11          11 percent complete
Iteration 12          12 percent complete
Iteration 13          13 percent complete
Iteration 14          14 percent complete
Iteration 15          15 percent complete
Iteration 16          16 percent complete
Iteration 17          17 percent complete
Iteration 18          18 percent complete
Iteration 19          19 percent complete
Iteration 20          20 percent complete
Iteration 21          21 percent complete
Iteration 22          22 percent complete
Iteration 23          23 percent complete
Iteration 24          24 percent complete
Iteration 25          25 percent complete
Iteration 26          26 percent complete
Iteration 27          27 percent complete
Iteration 28          28 percent complete
Iteration 29          29 percent complete
Iteration 30          30 percent complete
Iteration 31          31 percent complete
Iteration 32          32 percent complete
Iteration 33          33 percent complete
Iteration 34          34 percent complete
```

```
Iteration 35          35 percent complete
Iteration 36          36 percent complete
Iteration 37          37 percent complete
Iteration 38          38 percent complete
Iteration 39          39 percent complete
Iteration 40          40 percent complete
Iteration 41          41 percent complete
Iteration 42          42 percent complete
Iteration 43          43 percent complete
Iteration 44          44 percent complete
Iteration 45          45 percent complete
Iteration 46          46 percent complete
Iteration 47          47 percent complete
Iteration 48          48 percent complete
Iteration 49          49 percent complete
Iteration 50          50 percent complete
Iteration 51          51 percent complete
Iteration 52          52 percent complete
Iteration 53          53 percent complete
Iteration 54          54 percent complete
Iteration 55          55 percent complete
Iteration 56          56 percent complete
Iteration 57          57 percent complete
Iteration 58          58 percent complete
Iteration 59          59 percent complete
Iteration 60          60 percent complete
Iteration 61          61 percent complete
Iteration 62          62 percent complete
Iteration 63          63 percent complete
Iteration 64          64 percent complete
Iteration 65          65 percent complete
Iteration 66          66 percent complete
Iteration 67          67 percent complete
Iteration 68          68 percent complete
Iteration 69          69 percent complete
Iteration 70          70 percent complete
Iteration 71          71 percent complete
Iteration 72          72 percent complete
Iteration 73          73 percent complete
Iteration 74          74 percent complete
Iteration 75          75 percent complete
Iteration 76          76 percent complete
Iteration 77          77 percent complete
Iteration 78          78 percent complete
Iteration 79          79 percent complete
```

```
Iteration 80          80 percent complete
Iteration 81          81 percent complete
Iteration 82          82 percent complete
Iteration 83          83 percent complete
Iteration 84          84 percent complete
Iteration 85          85 percent complete
Iteration 86          86 percent complete
Iteration 87          87 percent complete
Iteration 88          88 percent complete
Iteration 89          89 percent complete
Iteration 90          90 percent complete
Iteration 91          91 percent complete
Iteration 92          92 percent complete
Iteration 93          93 percent complete
Iteration 94          94 percent complete
Iteration 95          95 percent complete
Iteration 96          96 percent complete
Iteration 97          97 percent complete
Iteration 98          98 percent complete
Iteration 99          99 percent complete
Iteration 100          100 percent complete

Call:
SSI.formula(formula = mathScore ~ readScore + scienceScore, data = tmp,     id = ID, k =

Coefficients:
Z.scores:  0.3702140 -0.5768174  0.1264285  0.8572412 -0.6106486
Percentiles: 0.644 0.282 0.550 0.804 0.271

> summary(result)

Call:
SSI.formula(formula = mathScore ~ readScore + scienceScore, data = tmp,
    id = ID, k = 20, na.action = na.omit)

Norms based on k =  20 similar students

> head(result$model.frame)

   mathScore  readScore scienceScore (id)    z.score percentile
1  0.6306851 -1.1240005   -0.3559542    1  0.3702140      0.644
2 -0.5206163 -0.4841261    0.2663100    2 -0.5768174      0.282
3 -0.1782192 -0.4419129    0.8635799    3  0.1264285      0.550
4  1.4947961 -0.1840422    0.9463427    4  0.8572412      0.804
5 -0.4547110 -0.6806347    0.5748169    5 -0.6106486      0.271
6  0.1933395 -0.6646900    0.9304420    6  0.7278767      0.767
```

The function returns a dataframe that has for each student included in the analysis a `z-score` and a percentile rank associated with that score. The help file describes the use of the arguments to this function. Here I note only that the values of k cannot be larger than the number of pairwise comparisons made.

16.10 Detecting Excessive Similarity in Student Response Patterns: The `cheat` function

This function requires that we provide two forms of data: a "raw" data file and a "key" file. The raw data are the student responses to each of the multiple choice items. For instance, if there are four options for an items (1,2,3,4), the data would indicate which option student i chose for item j. Second, the key file is a vector that contains the correct response for each item presented.

For example, if the correct response to item 1 is 4, then the value in the key file associated with this item would be a 1. It is best to illustrate with sample data as cnostructed below.

```
> NumStu <- 30
> NumItems <- 50
> dat <- matrix(0, nrow = NumStu, ncol = NumItems)
> set.seed(1234)
> for (i in 1:NumStu) {
+     dat[i, ] <- sample(1:4, NumItems, replace = TRUE)
+ }
> dat <- data.frame(dat)
> dat[(NumStu + 1), ] <- dat[NumStu, ]
> dat[(NumStu + 2), ] <- c(dat[(NumStu - 1), 1:25], dat[(NumStu -
+     2), 26:50])
> set.seed(1234)
> key <- sample(1:4, NumItems, replace = TRUE)
```

Note, the key is a vector and its length must equal the number of columns in the data. Like the other functions in this package, we also rely on the formula method here. However, before we use the `cheat` function we must first estimate the probability of choosing the wrong option for each test items. The function `wrongProb` can be used for this purpose as demonstrated below.

```
> ff <- as.formula(paste("~", paste(names(dat), collapse = "+")))
> mm <- wrongProb(ff, data = dat, key = key)
```

Now that we have the data, the key file, and the probability of choosing the wrong option, we simlpy call the cheat function as follows:

```
> (result <- cheat(ff, data = dat, key = key, wrongChoice = mm))

      Ncheat TotalCompare
           3          496
```

```
> summary(result)

 Number of Possible Cheatering Pairs: 3

 Possible Cheating Pairs: S28:32 S29:32 S30:31

 Number of Exact Matches: 29 31 50

 Observed Z Values: 4.69 5.26 10.97

 Critical Z: 4.11

 Expected Number of Matches: 13.71275 13.86224 14.42306

 Variance: 9.92999 9.991798 10.22801
```

The ouput from the summary tells us there are three students with response patterns too similar to have occured by random chance alone. The possible cheating pairs are `S28:32 S29:32 S30:31`. This means that the students in rows 28 and 32 seem to have copied from each other as did students in rows 29 and 32, and 30 and 31.

The output tells us that the observed number of exact matches between students 28 and 32 is 29. However, we would expect (from the statistical estimates) that they would only have 13.71 exact matches between them. In this case, the observed number of exact matches well exceeds the expected.

This function can be easily used to search through many classrooms (or any user defined groups) through the use of the subset command. For instance, say we are interested in examining all classrooms within a school district or a state. If multiple classrooms are to be examined, I recommend using the `wrongProb` function sing the population data file (all teachers) and then looping through all classrooms as follows:

```
> teachers <- unique(dat$TeacherName)
> result <- numeric(length(teachers))
> for (i in seq_along(teachers)) {
+     tmp <- cheat(~item1 + item2 + item3, dat, wrongChoice = mm,
+         key = key, rfa = "bsct", subset = TeacherName ==
+             teachers[i], na.action = NULL)
+     result[i] <- summary(tmp)$Ncheat
+ }
```

16.11 Equating with Stocking Lord

The `SL` function relies on the parameter estimates organized in a list in the same manner as what is required for the `irt.ability` function. Since we are equating two tests, we require two sets of item parameters organized as follows (for the 3PL):

```
> params1 <- list(`3pl` = list(a = c(0.4, 1.7, 1.2), b = c(-1.1,
+     0.9, 2.2), c = c(0.1, 0.2, 0.1)), gpcm = NULL)
> params2 <- list(`3pl` = list(a = c(0.5, 1.6, 1), b = c(-1.5,
+     0.5, 2), c = c(0.1, 0.2, 0.1)), gpcm = NULL)
```

With our item parameters organized in the list, we simply estimate the intercept and slope values as follows:

```
> SL(params1, params2, control = list(Q = 30, mu = 0, sigma = 1))

Stocking Lord Coefficients:
A parameter: 1.084583
B parameter: -0.5352729

Starting values from Mean/Sigma:
A parameter: 1.056315
B parameter: -0.3708769
```

This function puts the items from `params1` on the same scale as `params2`. That is, `params1` items would be rescaled using the linking constants. So, for example, if I were doing some vertical equating with this function and I want to link grade 3 and 4 with grade 3 as the base grade. I would apply the linking constants to the grade 4 items to place them on the grade 3 scale. In that case, `params1` would hold the grade 4 item parametersand `params2` would hold the grade 3 item parameters.

16.12 Levenshtein Distance Metric

The function `stringMatch` implements the procedures for matching two character strings as previously described. Suppose we simply want the unnormalized edit distance between two strings. we would simply use the function as:

```
> stringMatch("William Clinton", "Bill Clinton", normalize = "NO")

[1] 4
```

However, the LND (normalized) is often more useful for the application as we have described for merging records together. This is found using the argument `normalize = 'YES'` as:

```
> stringMatch("William Clinton", "Bill Clinton", normalize = "YES")

[1] 0.7333333
```

With pattern matching, it is sometimes needed to ignore differences in case, such as the following:

```
> stringMatch("Bill Clinton", "bill Clinton", normalize = "YES",
+     case.sensitive = FALSE)
```

```
[1] 1
```

However, we can also consider differences in case as follows:

```
> stringMatch("Bill Clinton", "bill Clinton", normalize = "YES",
+     case.sensitive = TRUE)
```

```
[1] 0.9166667
```

Now, suppose we have merged two data files together and have computed the LND for each record in the data comparing the names in the first dataset to the second. One question we may now ask is, "what value of the LND indicates a bad merge?" We can answer this question empirically and estimate the probability of a given LND.

These probabilities can be used in the same way researchers use alpha levels in other areas of scientific practice. That is, define the level of risk one is willing to accept, say .001, and find the LND associated with LND of .001. Then, use that LND as the cutpoint in the data and retain records only if they exceed that LND.

The stringency of the LND for merge validation depends in large part on the specific use of the data. For instance, if the results of any longitudinal analyses resulting from a data merge are to play a role in school- or teacher-specific decisions resulting in sanctions or rewards, then a stringent value of the LND should be preferred. Further research on this topic could enlighten practitioners on optimal LND selection for high-stakes environments.

We can use the `stringProbs` function to estimate these probabilities as follows:

```
> dat <- data.frame(fname1 = c("Joseph McCall", "Paul Jones",
+     "Larry Everett", "Sam Thompson", "Sally Fields",
+     "Doug Carter", "Bill Friendly", "Tom Davison", "Frank Mann",
+     "Mary Jones"), fname2 = c("Joe McCall", "Paul Jones",
+     "Barry Everett", "Samuel Thompson", "Sally Fields",
+     "Douglas Carter", "William Friend", "Tommy Davison",
+     "Franklin Mann", "Cary Jones"))
> stringProbs(dat, N = 5)
```

```
   Distance Prob CumProb
1      0.00 0.10    0.90
2      0.07 0.06    0.84
3      0.08 0.18    0.66
4      0.13 0.06    0.60
5      0.14 0.10    0.50
6      0.15 0.08    0.42
7      0.17 0.02    0.40
8      0.18 0.04    0.36
9      0.23 0.06    0.30
10     0.25 0.04    0.26
11     0.27 0.04    0.22
12     0.31 0.04    0.18
```

```
13      0.33 0.02    0.16
14      0.38 0.04    0.12
15      0.70 0.02    0.10
16      0.77 0.02    0.08
17      0.79 0.02    0.06
18      0.80 0.02    0.04
19      0.90 0.02    0.02
20      0.92 0.02    0.00
```