A Collection of Numerical Recipes Useful For Building Scalable Psychometric Applications

Harold Doran
hdoran@humrro.org
Human Resources Research Organization
Working Technical Paper

November 26, 2021

Abstract

Next generation psychometrics will inevitably leverage advancements made available with modern computing to create psychometric innovations. Realizing this future requires that psychometricians have the full suite of skills needed to create and deploy new innovations that can scale. Unfortunately, well-known numerical recipes for large-scale computing have not been fully embraced in psychometrics and this is a significant barrier to innovation. Commonly, textbook representations of mathematical solutions are confused to be the same computational strategy for implementation or trivial approximations are often used when faced with complex scenarios. Overcoming these limitations will require some focused conversations envisioning an advanced skill set for psychometricians involving numerical methods, computing, and software development. This work sets out to focus on only one of those components and a collection of numerical recipes is provided.

Keywords: computational psychometrics; item response theory; gaussian quadrature; linear models; numerical analysis; psychometric data scientist

Introduction

Psychometrics is evolving and increasingly becoming a complex and multidisciplinary field. The ideas and potential in front of next generation psychometricians presents a future for them to leverage advances in modern computing to build, scale, and implement redesigned psychometric models that better align with how individuals learn in modern society and demonstrate what they have learned in new ways (von Davier, Deonovic, Yudelson, Polyak, & Woo, 2019).

Stepping towards these advancements will inevitably require a concentrated focus on building a new type of psychometric innovator. This new innovator is of course trained in traditional measurement topics, but also has significant computational acumen coupled with the full suite of skills

needed to build and deploy software applications to make those ideas accessible for broad audiences. Consider, as an example, one computational psychometrics field guide proposed by ACT (ACTNext, 2019). The ideas in this guide have greater emphasis on topics that seemingly resemble pure computer science than topics currently emphasized in traditional measurement programs.

The future of psychometrics will be driven by the modern advantages provided by new technologies in at least three ways. First, some routine psychometric tasks can be reformulated to capture efficiencies so that the computing time is significantly reduced or provide avenues to directly tackle complex problems with deterministic approaches rather than reverting to simulation-based approximations (Lee, Leemaqz, & McLachlan, 2016). Second, psychometricians sufficiently familar with computational methods and software development will lead the advancement with new ideas and innovations by integrating psychometrics with machine and deep learning concepts. Third, those with a concentrated focus on psychometrics and computing will build scalable innovations that can manage the enormous amounts of available data and more easily distribute those ideas at scale using cloud-based infrastructures.

One possible negative side effect of modern computing is that it is almost too easy to rely on auto-scaling in cloud-based infrastructures or distributed computing concepts to split up tasks. These methods allow for a developer to easily scale up any computational task by quickly involving shared computing power, which inevitably involves greater cost. However, it is critically more important as a first step to consider the smallest and most manageable unit–the computational methods used in the underlying code to build the application. Then, good computational methods can be more readily scaled than expecting poor methods to be remedied by more powerful computing.

Numerical methods are the core at which ideas are implemented and psychometric innovators should have a portfolio of routine methods at the ready to support the development of new ideas. This will become even more critical as the demands of machine learning (ML) techniques take greater foothold in psychometrics with training data sets containing upwards of millions of observations to build classifiers and detect patterns. Unfortunately, efficient numerical methods are not widely embraced within the field of psychometrics and, perhaps, this is rooted in how we learn. One general observation is that textbooks commonly represent psychometric solutions with scalar-based expressions or, when matrices are displayed, they are typically conveyed in ways that conflate computational implementation with the notational expression. Notational representations should be understood as valuable for developing a conceptual understanding of the psychometric problem and to communicate the idea of the model or its parameterization. However, those same representations should not be interpreted as the same way to build a computational implementation, although it appears they often are.

Bates (2004) noted that the algebraic textbook solution for the least squares model is commonly used within R code and provided details for better computational implementation. Genz and Kass (1997) suggested that statisticians often ignore superior methods for high dimensional integration, a problem also described by Antal and Oranje (2007) in psychometric software applications finding that trivial rectangular rules are common even in the National Assessment of Educational Progress (Allen, Donoghue, & Schoeps, 2001) and commercially available psychometric software.

There are, however, many interesting conversations among psychometric innovators regarding the challenges in building scalable applications. Cai, Yang, and Hansen (2011) openly discuss the specific challenges between approaches for dealing with high dimensional integration problems as do the developers of the R package Dire (Doran, Bailey, Buehler, & joo Lee, 2021) for estimating

the marginal likelihood regression (Mislevy, 1984) which argues for a computational trade-off that may be realized by the simpler numerical integration rule relative to using a more sophisticated approach. Rosseel (2021) provides an impressive example that translates a complex problem into a feasible computational solution. This work makes good use of a standard inversion lemma to reduce the overhead associated with inverting large matrices and takes advantage of certain special structures and reduces the problem to a yield a simpler solution and is an exemplar of psychometric computing.

Notably, these conversations are typically focused on a specific application or problem. In contrast, the work presented here is intentionally problem agnostic with the goal of organizing a general class of useful recipes to be used as templates for creating new solutions. For this reason, this work is more focused on application and is therefore an organization of tested numerical approaches with the hope that they become more familiar to psychometricians. This work does not set out to formally prove theorems or engage in deriving estimators and proving their properties other than one simple proof in the appendix. Rather, it is an exploration of computational psychometrics considered through the lens of integrating psychological measurement with computational templates that become springboards to create new psychometric models that are computationally demanding. Consequently, the hope is that new psychometric innovations will emerge as psychometricians develop, implement, showcase, and advance new ideas rooted in sound computing methodologies.

It's important to acknowledge that numerical analysis techniques are situational and implemented as preference by the innovator suitable for a given condition. For instance, the examples described here make use of a certain decomposition; however, others may have legitimate reasons for others. Hence, there is no motivation to oversell these specific concepts as the most desirable for any scenario. In fact, that's exactly the opposite point. Instead, the underlying motivation is to encourage staring at a problem and finding a recipe through exploration and testing that yields a stable and reduceable structure to implement a complex procedure at scale. In other words, the idea proposed is that psychometricians need a toolkit allowing them to make a big problem small and the ideas presented here are generally useful, but not intended as the most ideal for every scenario.

Anecdotally, the work described here is a reflection of many conversations over the years answering the basic questions, "how do I write code for this?" or "why is my code taking hours (or days) to reach a stationary point?" For this reason, the hope is to broaden those conversations to a larger audience and provide what is hoped to be a useful resource for those interested in computing complex psychometric models. The recipes provided here are introduced in the framework of familiar and simple problems and the ideas are advanced sequentially to more complex scenarios. The work ends with three examples of methods not traditionally found in standard software simply to illustrate implementation of computational ideas and not to center attention on those specific methods.

Notation and Terminology

Standard notation is generally used throughout and the terms and notation are defined within each topic. For instance, matrices assume the general notation of uppercase bold, $\boldsymbol{A}$, determinants of matrices, $|\cdot|$, vectors are generally lowercase bold, $\boldsymbol{x}$, elements within sets $\{x_1, x_2, \ldots, x_n\}$, $\mathbb{E}(\cdot)$ is used for expected values, and functions of variables with a general form $f(\cdot)$. One convention

to clarify is the distinction between $g(\theta; \boldsymbol{\beta})$ and $g(\theta|\boldsymbol{\beta})$. The former is used to mean the density of the random variable at the value of θ with the parameters $\boldsymbol{\beta}$ and the latter is used to mean the conditional distribution of the random variable given the parameters.

Distinctions in terminology are made throughout using "algebraic" representation and "computational" representation to mean very different things. Specifically, "algebraic solution" is used to mean the way a solution to a problem is written in order to convey a concept. Instead, a "computational solution" is a numerical recipe that provides a stable way to implement a solution in software to estimate parameters that may not resemble the algebraic solution. The distinction between the two is a non-trivial point.

Introductory Concepts For Dealing With Correlated Data

Whitening Transformations for Correlated Variables

A concept first introduced and used throughout almost all subsequent recipes is referred to as the whitening transformation. Whitening is a process of transforming a collection of correlated vectors into a collection of uncorrelated vectors (Kessy, Lewin, & Strimmer, 2018) with the aid of a suitable decomposition. The process is critical to the solution of linear systems and for fast quadrature routines and is generally helpful for reducing complex problems into more computationally feasible solutions.

The concept is introduced as follows. Let $\boldsymbol{Y}$ be a matrix of correlated variables from a P-dimensional multivariate normal, $\boldsymbol{Y} \sim \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{\Sigma}$ is symmetric and positive definite. Now find a decomposition that satisfies $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}'$ where $\boldsymbol{L}$ is lower triangular known as the Cholesky factor and $\boldsymbol{L}'$ is its transpose. This provides that the change of variable transformation can be used as $\boldsymbol{X} = \boldsymbol{Y}\boldsymbol{L}'^{-1}$ so that the variables in the matrix $\boldsymbol{X}$ are "whitened" or "de-correlated". Note that $\boldsymbol{L}'^{-1}$ is used as the algebraic representation; finding the inverse of a triangular matrix is simple using forwards or backwards substitution. Notice this can also be used to create correlated vectors when the operation is reversed, $\boldsymbol{X}\boldsymbol{L}' = \boldsymbol{Y}$, which may be useful for data generation, simulation, or applications such as high dimensional integration.

There are many useful decompositions in numerical analysis (Searle, 1982; Zhang, 1999). However, the Cholesky is broadly helpful to speed computations and some work has demonstrated its potential advantage over others, such as the QR (Lira, Iyer, Trindade, & Howle, 2016) when used for solving normal equations. Some applications have fully adopted a sparse Cholesky decomposition (Davis & Hager, 2005) for linear mixed models (Bates, 2004b) with tremendous success. For these reasons, the Cholesky is the primary decomposition used throughout.

Multivariate Densities of Whitened Data

The multivariate normal density function has an important role in psychometric practice. It is possible to reduce this from a complex problem into separable parts and show how the multivariate density can be computed as a product of transformed univariate densities. This change of variable transformation has broader applications for psychometric computing in maximization methods or dealing with high-dimensional integration problems.

The general form of the P-dimensional multivariate normal for a non-diagonal covariance matrix $\boldsymbol{\Sigma}$ is

$$f_y(y_1, \ldots, y_p) = \frac{\exp(-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}))}{\sqrt{(2\pi)^p|\boldsymbol{\Sigma}|}} \tag{1}$$

If the mean vector is $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = diag(1, \ldots, 1)$, then the density of the multivariate normal is equal to the product of the marginal densities. Let φ denote the standard normal probability density function (p.d.f.), then

$$f_y(y_1, \ldots, y_p) = \prod_{i=1}^{p} \varphi(y_i) \tag{2}$$

Note that (1) and (2) are related only under the special case of uncorrelated variables with unit variance. With correlated variables, rewrite (1) as

$$f_y(y_1, \ldots, y_p) = \frac{\exp(-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{\mu})'\boldsymbol{L}^{-1}\boldsymbol{L}'^{-1}(\boldsymbol{y} - \boldsymbol{\mu}))}{\sqrt{(2\pi)^p|\boldsymbol{L}\boldsymbol{L}'|}}. \tag{3}$$

Now let $\boldsymbol{x}^{*'} = \boldsymbol{y}'\boldsymbol{L}^{-1}$ such that the elements of $\boldsymbol{x}^{*'} = \{x_1^*, \ldots, x_p^*\}$, then

$$f_y(y_1, \ldots, y_p) = \frac{\exp(-\frac{1}{2}\boldsymbol{x}^{*'}\boldsymbol{x}^*)}{\sqrt{(2\pi)^p|\boldsymbol{L}^2|}} \tag{4}$$

$$= |\boldsymbol{L}|^{-1}\left(\prod_{i=1}^{p} \varphi(x_i^*)\right) \tag{5}$$

This transform expresses (5) into the form of (2), thus showing how the multivariate density can be computed as a product of the (transformed) marginal densities.

Computational Concepts for Linear Systems

This section explores efficient computational recipes for linear models to create simple, reduceable structures to build scalable psychometric algorithms. It's best to begin with a simple, common problem–ordinary least squares (OLS), that will serve as a foundation and subsequently show how more complex problems can be reduced to the same OLS solution. While the examples are provided within the context of linear regressions, the same concepts are used widely across psychometric practice. For example, these same methods are often used in the expectation-maximimation (EM) algorithm during the M-step, in natural language processing problems, latent semantic analysis, solutions to factor analysis problems, in highly parameterized item calibrations problems using the newton-raphson algorithm, and within iteratively reweighted least solutions for generalized linear models. These are just a subset of the possible applications.

Ordinary Least Squares as a Unifying Framework

The standard representation of the linear least squares problem is via the following normal equations (McCulloch & Searle, 2001)

$$\boldsymbol{X'X\beta} = \boldsymbol{X'y} \tag{6}$$

where $\boldsymbol{X}$ is an $n \times p$ design matrix of full column rank, $\boldsymbol{\beta}$ is a $p \times 1$ vector of unknown parameters to be estimated, and $\boldsymbol{y}$ is an $n \times 1$ vector of observed outcomes. The algebraic representation in standard notation is shown having an invertable $\boldsymbol{X'X}$ with a solution written as

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{X'X})^{-1}\boldsymbol{X'y}. \tag{7}$$

The algebraic representation of the solution for the estimates of the fixed effects in Equation (7) is generally conflated with its computational solution. It is in fact unnecesary to actually invert $\boldsymbol{X'X}$; instead it is faster and more stable to find a decomposition for $\boldsymbol{X'X}$ and then solve triangular systems for the parameters rather than using matrix inversion.

One stable approach is to find the decomposition, $\boldsymbol{X'X} = \boldsymbol{LL'}$, and then rewrite (6) as $\boldsymbol{LL'\beta} = \boldsymbol{X'y}$. This expression provides a simple solution via two triangular systems requiring to first forwardsolve for $\boldsymbol{z}$ and then backsolve for $\boldsymbol{\beta}$ as

$$\boldsymbol{Lz} = \boldsymbol{X'y},\ \boldsymbol{L'\beta} = \boldsymbol{z}. \tag{8}$$

Obtaining least squares estimates is very simple. However, it serves as a baseline and a general unifying framework for computing linear systems as subsequently shown when more complex models can be rewritten into the simplified form of (6). In such cases, other models can be shown to inherit the simple solution shown in (8) even when they appear to be more complex, hence it unifies approaches to computing linear systems.

Generalized Least Squares in OLS Form

The generalized least squares (GLS) problem is an incrementally more advanced model than its OLS counterpart. The standard textbook normal equations for the GLS model is

$$\boldsymbol{X'\Sigma^{-1}X\beta} = \boldsymbol{X'\Sigma^{-1}y} \tag{9}$$

where $\boldsymbol{\Sigma} = \boldsymbol{LL'}$ is a symmetric $(n \times n)$ positive definite covariance matrix and all others matrices are as previously noted. The algebraic solution for the GLS approach is commonly shown to involve matrix inversions as follows

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{X'\Sigma^{-1}X})^{-1}\boldsymbol{X'\Sigma^{-1}y}. \tag{10}$$

This algebraic representation is conceptually useful, but does not make for an inefficient computational implementation. In particular, the solution to (10) implies inverting the matrix $\boldsymbol{\Sigma}$ and then again inverting $(\boldsymbol{X'\Sigma^{-1}X})$. The dimensions of $\boldsymbol{\Sigma}$ may very large as it is equal to the number of observations in $\boldsymbol{X}$ and the GLS problem often involves an iterative solution with elements of $\boldsymbol{\Sigma}$ changing over the iterations.

The concept of whitening can be used to rewrite the normal equations in (9) such that the data in $\boldsymbol{X}$ are "de-correlated" as

$$\boldsymbol{X'L^{-1}L'^{-1}X\beta} = \boldsymbol{X'L^{-1}L'^{-1}y} \tag{11}$$

$$\boldsymbol{X^{*'}X^{*}\beta} = \boldsymbol{X^{*'}y^{*}} \tag{12}$$

where $\boldsymbol{X}^{*} = \boldsymbol{XL}^{'-1}$ and $\boldsymbol{y}* = \boldsymbol{y'L'^{-1}}$. This whitening transform writes (9) into the form (6) so that it can inherit the same solution show in (8).

If (12) were used in an iterative process and elements of $\boldsymbol{\Sigma}$ were updated at each step, then recomputing its Cholesky factor repeatedly would layer additional expense. This is where numerical recipes become situational and we encourage staring at $\boldsymbol{\Sigma}$ to determine if it can be separated and reduced into a simple structure. We provide one such example in the section on linear mixed models where a very large matrix is repeatedly updated in an iterative algorithm and finding its inverse at each iteration is costly. Instead, a standard inversion lemma is used and it is extended to exploit some special characteristics of that matrix allowing a complicated problem to be reduced into a very trivial computation implementation.

Weighted Least Squares in OLS Form

The weighted least squares (WLS) problem is a special case of the GLS problem when the matrix $\boldsymbol{\Sigma}$ in the preceding example is diagonal. In this case, it is entirely unnecessary to actually create and store the entire matrix $\boldsymbol{\Sigma}$ and instead only its diagonal elements, v_{jj}, are needed. In this case there is a special matrix property providing that $\boldsymbol{X'\Sigma^{-1}X} = \boldsymbol{X'X^{*}}$ where is arrived at $\boldsymbol{X}^{*}$ via the elementwise calculation of $1/v_{jj}$ over each column of the matrix $\boldsymbol{X}$.

Performing this elementwise calculation to the matrix $\boldsymbol{X}$ on the left and right hand side of Equation 9 when $\boldsymbol{\Sigma}$ is diagonal yields

$$\boldsymbol{X'X^{*}\beta} = \boldsymbol{X^{*'}y}. \tag{13}$$

Equation (13) also is now written in the form of (6) and inherits the same computational solution.

Stochastic Gradient Descent for Linear Systems

The previously discussed methods show that linear models in the form of (6) inherit the general solution of (8). However, when the dimensions of the model matrix $\boldsymbol{X}$ become extremely large, then computing by (8) may prove challenging. Stochastic gradient descent (SGD) is a powerful approach for building highly scalable applications that reduces the computational burden associated with large data by simplifying it to an iterative solution passing over the gradient one sample at a time (Cizek & Cizkova, 2004; Shamir & Zhang, 2013).

While SGD is a general purpose optimization technique useful for minimization of any differentiable objective function, $\mathcal{J}(\boldsymbol{\theta})$, it is presented within the section on linear models as it provides a fast and clear pathway for many common psychometric solutions, such as with machine learning applications. It's conceivable that some training data sets used for ML could have hundreds of thousands or even millions of observations, in which case computing by (8) may be unrealistic.

Formally, let $\nabla \mathcal{J}_i(\boldsymbol{\theta})$ denote the gradient vector of the objective function with respect to the parameters to be minimized for an ith sample in the data. SGD performs the optimization by drawing samples from the observed data and repeatedly updating the gradient and the parameters at each iteration as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla \mathcal{J}_i(\boldsymbol{\theta}) \tag{14}$$

where $\alpha > 0$ is a learning rate parameter, $\boldsymbol{\theta}_t$ is the provisional value of the parameters at iteration t, and $\nabla \mathcal{J}_i(\boldsymbol{\theta})$ is the value of the gradient computed using sample i.

With respect to models assuming the form of (6), the gradient is $\nabla \mathcal{J}(\boldsymbol{\theta}) = 2N^{-1}(\boldsymbol{X}'\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}'\boldsymbol{y})$. However, very large dimensions in $\boldsymbol{X}$ make general optimization difficult with either general gradient descent or using the traditional closed form solution in (8). Instead, the iterative process outlined in Algorithm (1) allows for a large problem to be reduced simply.

Algorithm 1 Stochastic Gradient Descent

Input: Choose initial values for $\boldsymbol{\theta}$, $\nabla \mathcal{J}_i(\boldsymbol{\theta})$, and select a value for α.

1: Set $\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \alpha \nabla \mathcal{J}_i(\theta)$ at iteration t.
2: Sample $\boldsymbol{X}_i$, in $\boldsymbol{X}$ and its corresponding value, $\boldsymbol{y}_i$.
3: Compute $\nabla \mathcal{J}_i(\boldsymbol{\theta})$ using $\{\boldsymbol{X}_i, \boldsymbol{y}_i\}$.
4: Iterate between steps (1) and (3) until criterion for a stopping rule has been satisfied.

When $i = 1$ this process is termed stochastic gradient descent. However, the same iterative process can be used when $i > 1$ in small batches, termed mini-batch gradient descent. When $i = N$ with N being the total rows in $\boldsymbol{X}$, this is termed batch gradient descent.

As an aside, iterative algorithms need a stopping rule, often referred to as convergence criteria. In many respects, it's an arbitrary rule constructed for a given condition and there are multiple ways to achieve this outcome. For example, one might compute changes in the parameter estimates from iteration t to $t + 1$ and take the difference between them. Stopping might occur if no parameter differs by more than ϵ, a term set to be very small indicating the parameters are not changing enough from iteration to iteration any longer. Or, one might stop when differences in the log-likelihood differ by less than ϵ over iterations. In the SGD scenario, one might stop after reaching a fixed number of iterations or when the sum of the gradients squared is less than ϵ. No particular rule can be highly endorsed, although some scenarios are susceptible to being stuck in a saddle point.

Linear Mixed Models with Henderson's Method

The linear mixed model is generally written with the form $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\boldsymbol{u} + \boldsymbol{e}$ (Laird & Ware, 1982) and is often presented as having the normal equations (McLean, Sanders, & Stroup, 1991)

$$\begin{bmatrix} \boldsymbol{X}'\boldsymbol{\Omega}^{-1}\boldsymbol{X} & \boldsymbol{X}'\boldsymbol{\Omega}^{-1}\boldsymbol{Z} \\ \boldsymbol{Z}'\boldsymbol{\Omega}^{-1}\boldsymbol{X} & \boldsymbol{Z}'\boldsymbol{\Omega}^{-1}\boldsymbol{Z} + \boldsymbol{G}^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\beta} \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}'\boldsymbol{\Omega}^{-1}\boldsymbol{y} \\ \boldsymbol{Z}\boldsymbol{\Omega}^{-1}\boldsymbol{y} \end{bmatrix} \tag{15}$$

where $\boldsymbol{y}$ is an $n \times 1$ vector of outcomes, $\boldsymbol{X}$ is an $n \times p$ model matrix for the fixed effects, $\boldsymbol{\beta}$ is a $p \times 1$ vector of estimates for the fixed effects, $\boldsymbol{Z} = [\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_Q]$ is an $n \times Q$ model matrix for the random effects, $\boldsymbol{u}' = [\boldsymbol{u}'_1, \boldsymbol{u}'_2, \ldots, \boldsymbol{u}'_Q]$ is the vector of random effects, and $\boldsymbol{e}$ is the residual

error term. The matrices $\boldsymbol{\Omega}$ and $\boldsymbol{G}$ can have many forms depending on the structure of the model and are developed below for one variant of the linear mixed model.

Algorithm 2 Henderson Mixed Model Sketch

Input: Create starting values for σ^2_ϵ and $\sigma^2_q \; \forall \; q$.

1: Construct $\boldsymbol{\Omega} = \sigma^2_\epsilon \boldsymbol{I}_n$ and $\boldsymbol{G} = diag(\{\sigma^2_1, \ldots, \sigma^2_1\}, \{\sigma^2_2, \ldots, \sigma^2_2\}, \ldots, \{\sigma^2_Q, \ldots, \sigma^2_Q\})$. The length of the diagonal element for block q is equal to the number of columns in its corresponding matrix, $\boldsymbol{Z}_q$.
2: Solve the linear system for $\boldsymbol{\beta}$ and $\boldsymbol{u}$.
3: Update the values of the variances of the random effects including σ^2_ϵ and σ^2_q.
4: Iterate between steps (1) and (3) until criterion for a stopping rule has been satisfied.

Given the representation of (15) there is a temptation to assume a naive solution to the linear system using matrix inversion of the leftmost matrix. This is in fact how the SAS software documents the solution to the mixed model in its technical manual (SAS Documentation 14.2, n.d.). This may be perhaps just the algebraic representation and not actually the computational implementation. Instead, the computational solution can be written to resemble the solutions to linear systems previously described and a sketch of the iterative process is provided in Algorithm (2). Begin by writing

$$\boldsymbol{LL}' = \begin{bmatrix} \boldsymbol{X}'\boldsymbol{\Omega}^{-1}\boldsymbol{X} & \boldsymbol{X}'\boldsymbol{\Omega}^{-1}\boldsymbol{Z} \\ \boldsymbol{Z}'\boldsymbol{\Omega}^{-1}\boldsymbol{X} & \boldsymbol{Z}'\boldsymbol{\Omega}^{-1}\boldsymbol{Z} + \boldsymbol{G}^{-1} \end{bmatrix}, \; \boldsymbol{y}^* = \begin{bmatrix} \boldsymbol{X}'\boldsymbol{\Omega}^{-1}\boldsymbol{y} \\ \boldsymbol{Z}\boldsymbol{\Omega}^{-1}\boldsymbol{y} \end{bmatrix} \tag{16}$$

where elementwise calculation can be used for the diagonal matrices (e.g., $\boldsymbol{\Omega}^{-1}\boldsymbol{X}$) as presented in the WLS section and $\boldsymbol{\Theta}' = [\boldsymbol{\beta}' \; \boldsymbol{u}']$. The model can now be written in the form $\boldsymbol{LL}'\boldsymbol{\Theta}' = \boldsymbol{y}^*$ sharing the representation of the OLS form and shares the same solution using (8). First solve $\boldsymbol{Lz} = \boldsymbol{y}^*$ for $\boldsymbol{z}$ and then solve $\boldsymbol{L}'\boldsymbol{\Theta} = \boldsymbol{z}$ for $\boldsymbol{\Theta}$. Of course, this is only one iteration and solutions for the mixed model iterate with updated values for the variance components. Note, that when all elements of the matrix $\boldsymbol{G} = \boldsymbol{0}$, this reduces to the partitioned fixed effects model.

Variance Estimation with Woodbury Identity and Decompositions for Matrix Inversions

Obtaining updated values for the variance components at each iteration of the algorithm can be estimated by first computing (McCulloch & Searle, 2001)

$$\boldsymbol{T} = \left[\boldsymbol{I} + \left(\boldsymbol{Z}'\boldsymbol{Z} - \boldsymbol{Z}'\boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Z}\right)\boldsymbol{D}\right]^{-1} \tag{17}$$

where $\boldsymbol{I}$ is an identity matrix with dimensions equal to the model matrix for the random effects and $\boldsymbol{D} = \sigma^{-2}_{e,t-1}\boldsymbol{G}$ and all other matrices are as above. The residual variance at iteration $t-1$ is

$$\sigma^2_{e,t-1} = \frac{\boldsymbol{y}'\boldsymbol{e}}{N-p} \tag{18}$$

and the estimate of the variance at level q is

$$\sigma_q^2 = \frac{\widetilde{\boldsymbol{u}}'_{q,t-1}\widetilde{\boldsymbol{u}}_{q,t-1}}{m_q - tr(\boldsymbol{T}_q)} \tag{19}$$

where $\widetilde{\boldsymbol{u}}_q$ are the predictions of the random effects at level q, m_q are the number of units in level q, and $tr(\boldsymbol{T}_q)$ is the trace of the matrix $\boldsymbol{T}_q$, which is the block of the matrix $\boldsymbol{T}$ corresponding to the qth block.

The challenge here is that the square matrix $\boldsymbol{T}$ can be very large. As a result, it is extremely expensive to find its inverse inside an iterative problem, thus rendering this particular approach computationally prohibitive unless a reduceable structure can be identified. In some cases, there is an alternative method for taking the inverse of a matrix with this structure via the Woodbury Matrix Identity (Woodbury, 1950) that provides a very efficient route for computation. First, Equation (17) is rewritten as

$$\boldsymbol{T} = \left(\boldsymbol{I} + \boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D} - \boldsymbol{Z}'\boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Z}\boldsymbol{D}\right)^{-1} \tag{20}$$

where $\boldsymbol{D}$ is a diagonal matrix in the case of random intercepts and within each block q is constant along the diagonal. The Woodbury Identity provides that $\boldsymbol{T}$ can be computed as

$$\boldsymbol{T} = \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{B}\left(\boldsymbol{P} - \boldsymbol{B}'\boldsymbol{D}\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1}\boldsymbol{B}'\boldsymbol{D}\boldsymbol{A}^{-1} \tag{21}$$

where $\boldsymbol{A} = \boldsymbol{I} + \boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D}$, $\boldsymbol{B} = \boldsymbol{Z}'\boldsymbol{X}$, and $\boldsymbol{P} = \boldsymbol{X}'\boldsymbol{X}$. Now using (21) $\boldsymbol{T}$ can be computed by separately finding the inverse of $\boldsymbol{A}$ and then also of $\boldsymbol{P} - \boldsymbol{B}'\boldsymbol{D}\boldsymbol{A}^{-1}\boldsymbol{B}$. The latter of the two has very small dimensions, only $p \times p$ where p is the number of fixed effects in the model. The portion $\boldsymbol{I} + \boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D}$ is a bit more cumbersome because the dimensions of the model matrix for the random effects is $Q \times Q$, where Q is the number of columns in the model matrix for the random effects which can be extremely large.

The components $\boldsymbol{I}$ and $\boldsymbol{Z}'\boldsymbol{Z}$ are fixed and never change over the iterative process, only $\boldsymbol{D}$ is changing at each iteration. For nested random effects, $\boldsymbol{Z}'\boldsymbol{Z}$ is block diagonal but it has no special structure when the random effects are fully or partially crossed. This fact is motivation to explore if the special structures can be exploited in order to more easily find $\boldsymbol{A}^{-1}$ given that it will be computed many times. Of course, it is a well-known result that the sum of inverses is not equal to the inverse of the sum. However, $\boldsymbol{I}$ and $\boldsymbol{D}$ are both diagonal within a block of q for nested random effects and have constant values along the diagonal. These structures make it possible to find $\boldsymbol{A}^{-1}$ in a trivial way for the linear mixed model when dealing with nested random effects. Because $(\boldsymbol{Z}'\boldsymbol{Z})^{-1}$ is block-diagonal in this case, its inverse is found one block at a time.

It's pointed out here that the following works in the special case of nested random effects for random intercepts and is not globally generalizable (see appendix). However, the emphasis in this work is to identify special structures for a given scenario and exploit them to reduce computational burden. The inverse of the sum provided here may also have broader applications in other types of problems, such as when repeatedly finding the inverse of a Hessian matrix in iterative maximization problems, and so it is a motivating use case to document the condition.

For convenience, the subscript on $\boldsymbol{Z}$ is dropped, but note that this performed on each block, or perhaps on large $\boldsymbol{Z}$, when it too has a special structure. First, precompute and store the eigendecomposition for $\boldsymbol{Z}'\boldsymbol{Z}$, which is the most expensive part of the calculation. Then, all subsequent iterations treat the eigenvectors as fixed and are reused and the eigenvalues are updated repeatedly

with virtually no computational overhead. The inverse of the sum is then simply the inverse of the eigenvalues as

$$(\boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D} + \boldsymbol{I})^{-1} = \boldsymbol{A}^{-1} = \boldsymbol{Q}_1\boldsymbol{\lambda}^{*-1}\boldsymbol{Q}_1' \tag{22}$$

where $\boldsymbol{Z}'\boldsymbol{Z} = \boldsymbol{Q}_1\boldsymbol{\lambda}_1\boldsymbol{Q}_1'$ and $\boldsymbol{\lambda}^* = diag\{\lambda_{11}d + 1, \lambda_{12}d + 1, \ldots, \lambda_{n1}d + 1\}$. Proof for (22) is provided in the appendix. Now $\boldsymbol{A}^{-1}$ is obtained in a much less expensive way by reusing, $\boldsymbol{Q}_1$, and simply updating $\boldsymbol{\lambda}^*$ at each iteration, which is a nominal task given that it is diagonal. So, pulling together the Woodbury Identity and the decomposition derived for Equation (22), the inverse of the very big matrix $\boldsymbol{T}$ is easily computed over and over again with little overhead.

Scalable Methods for Integration

Psychometricians routinely encounter challenging marginal likelihood problems or posterior distributions that require numerical integration methods. These methods approximate difficult integrals by summing over a finite number of points. Integrals with a gaussian factor are rather common in psychometric applications and, as such, the Gauss Hermite rule may be used for integrals of the following form over the domain $(-\infty, \infty)$

$$\int f(\theta)e^{-\theta^2}d\theta \approx \sum_{q=1}^{Q} f(\theta_q)w_q \tag{23}$$

where summation is over $\mathcal{P} = \{\theta_q, w_q\}_{q=1}^{Q}$, the so called nodes, θ_q, and weights, w_q, for $q = 1, 2, \ldots, Q$. The choice of nodes and weights in $\mathcal{P}$ is optimal under the Gauss Hermite rule and will generally outperform other methods (Quarteroni, Saleri, & Gervasio, 2010). Unfortunately, psychometric practice seems to have codified the basic rectangular rule with fixed, equally spaced points from the domain of $(-4, 4)$ (Bock & Mislevy, 1982) despite its serious limitations in most cases for at least two reasons. First, the equally spaced points may not be sampled in the best region of the integrand and second it lacks the computational advantage of precision until an extremely large number of fixed points are used. Building scalable applications requires approaches with minimal computational overhead that achieve precision rapidly.

One Dimensional Gaussian Quadrature

Conveniently, psychometricians encounter scenarios where the weight function is a standard normal density

$$\int f(\theta)\frac{e^{-\theta^2/2}}{\sqrt{2\pi}}d\theta \approx \pi^{-1/2}\sum_{q=1}^{Q} f(\sqrt{2}\boldsymbol{L}\theta_q + \widetilde{\theta})w_q \tag{24}$$

where $\widetilde{\theta}$ is a location parameter for centering (Liu & Pierce, 1994)[1]. The approximation in (24) implements a transformation on θ from (23) and adapts to the best region of the integrand–hence it is the adaptive Gauss Hermite rule. The expression in (24) suggests $\widetilde{\theta}$ and values to construct

[1]The $\boldsymbol{L}$ in (24) is actually a scalar, σ, but is left as a matrix so it appears as a special case of (26).

$\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}'$ are known, or at least approximately known, yet those are the latent quantities we are trying to estimate. We return to an approach to consider for replacing these with provisional values in a later section.

Multivariate Gaussian Quadrature

The one dimensional Gauss Hermite rule can be extended to the case for integrals of the general form in multiple dimensions (Chowdhary, Salloum, Debusschere, & Larson, 2015; Jäckel, 2005; Judd, Maliar, & Maliar, 2011; Stringer, 2021)

$$\int \dots \int f(\boldsymbol{\theta})\psi(\boldsymbol{\theta})d\boldsymbol{\theta}. \tag{25}$$

where $\boldsymbol{\theta} = \{\theta_1, \dots, \theta_p\}'$ and continue with the assumption $\psi(\boldsymbol{\theta})$ is multivariate normal. The integral in (25) is often intractable and requires approximation. This may generally be approximated with nodes rotated to reflect a covariance structure and possibly shifted over a mean. That rotation can be implemented with the Cholesky factor of a covariance matrix as (Jäckel, 2005; Judd et al., 2011; Stringer, 2021)

$$\int \dots \int f(\boldsymbol{\theta})\psi(\boldsymbol{\theta})d\boldsymbol{\theta} \approx \pi^{-P/2} \sum_{q_1=1}^{Q} \cdots \sum_{q_p=1}^{Q} f(\sqrt{2}\boldsymbol{L}(\theta_{q1}, \dots, \theta_{qp})' + (\widetilde{\theta}_{q1}, \dots, \widetilde{\theta}_{qp})')w_q^* \tag{26}$$

where θ_{qp} is node $q = 1, 2, \dots, Q$ in dimension $p = 1, 2, \dots P$, w_{qp} is the corresponding weight, $w_q^* = \prod_{p=1}^{P} w_{qp}$, and $\widetilde{\theta}_{qp}$ is used to mean some points around which the nodes are centered. The nodes are expansions of the points in the one-dimensional set, $\mathcal{P}$, via a cartesian product rule to form a multivariate grid, $\boldsymbol{G}$, with dimensions $Q^P \times P$. Hence, the qth row of $\boldsymbol{G}$ is $\boldsymbol{g}_q = (\theta_{q1}, \dots, \theta_{qp})$. Note that (24) is simply a special case of (26), hence no new concepts are needed for implementation beyond what has been described for the one-dimensional scenario.

The extremely large number of quadrature points is what renders this approach challenging as, hueristicaly, this is effectively a two step process. Step 1 requires evaluating the function $f(\cdot)$ over all Q^P nodes and then step 2 is the summation of (26) making use of those values over all rows of the grid $\boldsymbol{G}$. Even with a modest selection for Q, the exponential growth in the function evaluations quickly becomes large with increasing P. Hence, we might explore ways in which we evaluate $f(\cdot)$ as few times as possible and whether fewer function evaluations than $Q^P \times P$ can be used. These options are explored in the applications section.

General Purpose Optimization Tools

Perhaps the most pervasive need for psychometric application building is a collection of tools for optimizing likelihoods. The likelihood theory of inference is well-developed (King, 1998) and found across all corners of psychometric practice. The common challenge is that likelihood functions for psychometric problems rarely have analytic forms and so iterative numerical approaches are used for maximum likelihood estimation (MLE). Two methods are explored here with a reminder that the stochastic gradient descent method is also a general purpose tool, but was presented within the section on linear models.

Maximization with Newton-Raphson

Optimization problems in psychometrics generally involve a multi-parameter likelihood function intending to simultaneously maximize all parameters. In this instance, let $\mathcal{L}(\boldsymbol{\theta})$ denote a twice differentiable likelihood function with respect to the parameters $\boldsymbol{\theta}$ with gradient $\nabla\mathcal{L}(\boldsymbol{\theta})$ and Hessian, $\mathcal{H}(\boldsymbol{\theta})$. The Newton-Raphson procedure maximizes $\mathcal{L}(\boldsymbol{\theta})$ by choosing initial starting values for $\boldsymbol{\theta}$ and then iteratively updating the parameters via

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathcal{H}_t(\boldsymbol{\theta})^{-1}\nabla\mathcal{L}_t(\boldsymbol{\theta}) \tag{27}$$

where the subscript t denotes the values of the gradient and Hessian at iteration t and continues until a convergence criterion is reached. The notation, $\mathcal{H}_t(\boldsymbol{\theta})^{-1}$, implies inversion of this matrix at each iteration. However, as previously discussed, the operation should be performed using a numerically stable decomposition. Additionally, because $\mathcal{H}_t(\boldsymbol{\theta})^{-1}$ is found iteratively and may be expensive, it may be feasible to find a trivial way to perform this operation similar to the eigen-decomposition method in Equation (38) provided in the appendix that takes advantage of precomputing and reusing certain components. While the method in the appendix is unique to the mixed model scenario, it is intended to be a concept that can be generalized to other situations where a matrix inverse computed iteratively can be made less expensive. Quasi-Newton methods, such as the BFGS algorithm (Fletcher, 1987), are other less expensive options that are quite simple to implement and avoid directly computing $\mathcal{H}_t(\boldsymbol{\theta})^{-1}$, which may be difficult in some scenarios.

Expectation-Maximization Algorithm

There is no approach more ubiquitous across psychometrics than the expectation-maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977). The EM algorithm is based on the idea that there exists some complete data, $\boldsymbol{z} = \{\boldsymbol{x}, \boldsymbol{\eta}\}$, and a complete data likelihood, $\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}) = \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{\eta})$, that could be maximized with respect to $\boldsymbol{\theta}$ if the complete data were observed. The problem is that only $\boldsymbol{x}$ is observed and $\boldsymbol{\eta}$ are latent; hence the values of $\boldsymbol{\eta}$ represent missing data and so maximizing $\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z})$ cannot easily occur.

The EM algorithm ameliorates the missing data problem by using provisional, or conditional expected values, of $\boldsymbol{\eta}$ and then maximizing the complete data likelihood. Algorithm (3) outlines the alternating steps used to implement EM.

Algorithm 3 Expectation Maximization Algorithm

Input: Create starting values for $\boldsymbol{\theta}$.

1: E-step is to compute expected values given the observed data and the provisional parameter estimate, $Q(\boldsymbol{\theta}; \boldsymbol{\theta}_t) = \mathbb{E}[\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{\eta})|\boldsymbol{x}, \boldsymbol{\theta}_t]$.
2: M-step is to find $\operatorname{argmax}_{\boldsymbol{\theta}\in\boldsymbol{\Omega}} Q(\boldsymbol{\theta}; \boldsymbol{\theta}_t)$ where $\boldsymbol{\theta}$ is in the parameter space $\boldsymbol{\Omega}$.
3: Iterate between steps (1) and (2) until criterion for a stopping rule has been satisfied.

There are many examples of the EM algorithm in the psychometric literature for interested readers to further explore (Hsu, Ackerman, & Fan, 1999). One very simple example that can be quickly illustrated is to reconsider estimating the linear mixed model of the form $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\boldsymbol{u} + \boldsymbol{e}$. Here, the complete data are $\{\boldsymbol{y}, \boldsymbol{u}\}$ and if fully observed, the parameters of the mixed model

could be found using the MLE variants of Equations (18) and (19) for the variance components and Equation (8) for estimates of the fixed effects[2]. The random effects are in reality missing, but the joint distribution of the complete data, $\{\boldsymbol{y}, \boldsymbol{u}\}$, is assumed multivariate normal and standard distribution theory for the multivariate normal provides that $\mathbb{E}(\boldsymbol{u}|\boldsymbol{y})$ can be easily determined (McCulloch & Searle, 2001) and used to update a new value for the M step.

There are some occasions where the M-step may not have a closed form or data might be so large the maximization step could be daunting even when working values for the missing data are easily obtained. One possible interesting variant of the EM algorithm is to use SGD in the M-step to manage a large data problem. For example, in the case of the mixed model above, assume $\mathbb{E}(\boldsymbol{u}|\boldsymbol{y})$ is available, but maximization via (8) is hard because the dimensions of the model matrix $\boldsymbol{X}$ are extremely large. Instead of computing the M-step by (8), the SGD could be used in the M-step as a way to manage the process allowing computation to be highly scalable. Other occasions may arise, for instance, in generalized linear mixed models that involve an integral for the M-step and one of the tools discussed above for numerically approximating that integral might be used. In other cases, maximization may require an iterative solution, thus pulling in the Newton-Raphson procedure as a way to solve the M-step.

Applications to Psychometric Problems

The previously discussed methods are computational templates that can be used to build and scale psychometric applications. Three examples are provided here with supporting code in the appendix to bring some life to those concepts. The intent is not to center attention on these specific examples; rather these examples are used because they are approaches that typically require computing something complex and are not routinely found in standard software. For this reason, they are attractive for exploring computationally efficient approaches.

Example 1: Estimating the Error in Variables Linear Model

Estimates of examinee performance from a psychometric instrument are noisy reflections of a true, latent trait. The observed measures are subsequently used in various ways and often used as inputs in regression models of some form. When using these observed estimates, the assumptions of the Gauss-Markov theorem do not hold and the parameter estimates are then inconsistent unless corrections are made (Doran, 2014; Lockwood & McCaffrey, 2020; Nab, van Smeden, Keogh, & Groenwold, 2021).

These models are chosen as an example because with different assumptions about the nature of the measurement error (e.g., homoscedastic versus heteroscedastic), the models may take different forms and custom software is often written to obtain parameter estimates. One such example is the `mecor` (Nab et al., 2021) package in R and a code review of its implementation shows that the algebraic representation and computational representation are one in the same.

One form of the linear EiV regression is commonly shown to be (StataCorp, n.d.)

$$(\boldsymbol{X}'\boldsymbol{X} - \boldsymbol{S})\boldsymbol{\beta} = \boldsymbol{X}\boldsymbol{y} \tag{28}$$

[2]If the random effects were observed, the least squares solution is justified as $\boldsymbol{y} - \boldsymbol{Z}\boldsymbol{u} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{e}$

where, in the case of known estimates for the reliability, the matrix $\boldsymbol{S} = diag(N(1-r_1)\sigma_1^2, \ldots, N(1-r_j)\sigma_j^2)$ where r_j is the reliability of the jth variable, N is the number of individuals, and σ_j^2 is the sample variance of the jth variable in the model matrix $\boldsymbol{X}$. Now, find $\boldsymbol{X}'\boldsymbol{X} - \boldsymbol{S} = \boldsymbol{L}\boldsymbol{L}'$ and obtain estimates using (8). Code to implement a simple error-in-variables model with known reliability is provided in the appendix as an example. It's useful to note that (28) can be extended for the linear mixed model and the computational approach previously shown can be applied.

Example 2: Estimating a High-Dimensional EAP

Suppose we are interested in estimating the ability of an examinee using a P-dimensional EAP estimator. This is chosen as an example because it is a desirable estimator that continues to be challenging for psychometricians to implement (Chalmers, 2012; Ferrando & Lorenzo-Seva, 2016). Let the expected value of the posterior be

$$\widehat{\boldsymbol{\theta}} = \frac{\int \cdots \int \boldsymbol{\theta}\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z})\psi(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})d\boldsymbol{\theta}}{\int \cdots \int \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z})\psi(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})d\boldsymbol{\theta}} \tag{29}$$

where $\psi(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the multivariate normal density function and the likelihood function (see appendix) is a product of domain-specific likelihoods as $\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}) = \prod_{p=1}^{P} \mathcal{L}(\theta_p; \boldsymbol{z}_p)$. Individuals are assumed to have some idiosyncratic ability contributing to their response on each dimension and that some test items are uniquely associated with dimension 1 (θ_1), others with dimension 2 (θ_2), and so on for the Pth dimension, $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_P)'$.

The likelihood function and its corresponding population distribution in (29) are not in the same parametric family, as such the integral cannot be evaluated analytically. Here three options are explored that make for possible implementation that vary in the number of times the function $\mathcal{L}(\theta; \boldsymbol{z})$ is computed. Once the likelihood is evaluated, then step 2 is effectively direct implementation of the sum over all rows of $\boldsymbol{G}$. Sparse grids or "pruning" some values are options for reducing the number of points over which the function is evaluated, but those are not explored here.

The first canonical option is to use (26) directly. This involves using the Cholesky factor to adjust the nodes to absorb the population covariance structure and then, possibly shift the nodes over a new centered point. This works quite well, and in fact is implemented in the code in the appendix. This however, has one downside worth noting that makes this computationally somewhat less attractive.

Initially (before rotating with the Cholesky factor), the nodes in $\boldsymbol{G}$ are patterned such that marginally (i.e., each column in $\boldsymbol{G}$) the Q unique nodes are repeated over and over. When the rows of this matrix are rotated via $\boldsymbol{L}(\theta_{q1}, \ldots, \theta_{qp})'$, then the uniqueness is lost exponentially over each column. That is, column 1 has Q unique nodes repeated, column 2 has only Q^2 unique nodes, column 3 would have Q^3 unique nodes and so on up to the Pth column.

This reduced pattern of unique nodes creates a larger burden in that we must compute the values of the likelihood at each node and that IRT likelihood is somewhat expensive as it involves computing exponents. In column 1, it is feasible to compute the value of the likelihood at the Q unique points and then simply copy those values into the other positions in column 1 where the same value of the node appears. That is, we would compute the likelihood only Q times and then recycle those values by copying them into other positions where the same node appears in the grid. In column 2, we must compute the likelihood at all Q^2 unique nodes and then the values could then

be copied into the positions of this column where the same node appears again. The upside here is that the method yields a good approximation of the integral. The downside is the computational expense is very large. In this case, we would evaluate the likelihood $Q + Q^2 + Q^3, \ldots, Q^P$ times.

A second option for computing this integral could be to use a stochastic quadrature routine. In this process the nodes are random draws from $\mathcal{N}_p(\boldsymbol{M}, \boldsymbol{I}_P)$, where $\boldsymbol{I}_P$ is a P-dimensional identity matrix again creating the grid $\boldsymbol{G}$, but in this instance it would have dimensions $R \times P$ where R is the number of random variates chosen and it is not an exponential function of R. In this manner, the variates in $\boldsymbol{G}$ would reflect the population covariance after being premultiplied by the Cholesky factor (see introductory section) when the summation in (26) is used to evaluate the integral. This stochastic approach makes the computation reasonable and its precision is highly dependent on the law of large numbers such that $\widehat{\theta}_p \xrightarrow{a.s.} \theta_p$ as $R \to \infty$[3]. In this case, we would evaluate the likelihood $R \times P$ times.

Perhaps a third and seemingly very rapid approach can be considered. First, rewrite the integral within (29)

$$\int \cdots \int \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}) \psi(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{\theta} = \int \cdots \int \frac{\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}) \psi(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\widetilde{\psi}(\boldsymbol{\theta}; \widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\Sigma}})} \widetilde{\psi}(\boldsymbol{\theta}; \widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\Sigma}}) d\boldsymbol{\theta} \tag{30}$$

and then inserting back into (29) gives the approximation

$$\widehat{\theta}_p = \frac{\sum_{q=1,1}^{Q} \cdots \sum_{q=p,1}^{Q} \theta_{p,q} \mathcal{L}(\theta_{1,q}; \boldsymbol{z}_1) \mathcal{L}(\theta_{2,q}; \boldsymbol{z}_2) \cdots \mathcal{L}(\theta_{P,q}; \boldsymbol{z}_p) \phi(\boldsymbol{\theta}_q; \boldsymbol{\mu}', \boldsymbol{\Sigma}') w_q^*}{\sum_{q=1,1}^{Q} \cdots \sum_{q=p,1}^{Q} \mathcal{L}(\theta_{1,q}; \boldsymbol{z}_1) \mathcal{L}(\theta_{2,q}; \boldsymbol{z}_2) \cdots \mathcal{L}(\theta_{P,q}; \boldsymbol{z}_p) \phi(\boldsymbol{\theta}_q; \boldsymbol{\mu}', \boldsymbol{\Sigma}') w_q^*} \tag{31}$$

where

$$\phi(\boldsymbol{\theta}_q; \boldsymbol{\mu}', \boldsymbol{\Sigma}') = \frac{\psi(\boldsymbol{\theta}_q; \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\widetilde{\psi}(\boldsymbol{\theta}_q; \widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\Sigma}})} \tag{32}$$

where $\widetilde{\psi}(\cdot)$ is a proposal distribution. This expression resembles an importance sampling approach (Gelman, Carlin, Stern, & Rubin, 2004) and has been termed importance Gauss Hermite (IGH) method (Elvira, Martino, & Closas, 2021) and other applications have used similar ideas (Antal & Oranje, 2007; Tuerlinckx, Rijmen, Verbeke, & De Boeck, 2006; Pinheiro & Bates, 1994). The value of the IGH approach is that the weights are modified to have greater importance for certain values of the integrand, in which case fewer quadrature points may be needed to evaluate the integral. The trade-off is that some calculations involving the densities in $\phi(\cdot)$ is required, though that additional cost is seemingly smaller than if more quadrature points were used and importance-based methods may be valuable for reducing computational burden (Ackerberg, 2009).

The additional benefit is that the original (unrotated) Gauss Hermite nodes are used in (31). We can exploit the uniqueness in the repeating values of the nodes in each column and compute the value of the likelihood only Q times in each dimension and then copy those values into other positions of the column where they repeat. As a result, the likelihood is evaluated only $Q \times P$ times and not $Q^P \times P$ times! Assuming $Q < R$, this approach evaluates the likelihood the smallest number of times of all three cases.

[3]The asymptotics here also depend on the number of test items being very large and not only the number of variates used for evaluation

Naturally, the question of what values are used in $\widetilde{\psi}(\cdot)$ needs clarification. This choice is what has the largest impact relative to the computational burden, yet unfortunately, some experimentation is needed for choosing these values. The implementation provided in the appendix uses $\widetilde{\psi}(\boldsymbol{\theta}; \mathbf{0}, \boldsymbol{I})$ as a template.

One final consideration for reducing some computational burden in problems of this nature might be to use an iterative process as suggested by Naylor and Smith (1982). If values for the parameters of interest were known, or at least approximately known, the nodes could be recentered via the transformation $\sqrt{2}\theta_{p,q}\sigma_p + \widehat{\theta}_p$ and take another pass over the integral with the transformed nodes and weights at an increased number of quadrature points for precision could be used.

R code for implementing this problem is provided in the appendix. It is a generalizable function that can evaluate any P-dimensional problem. Some additional computational benefits could easily be realized in this code. For instance, computing the likelihood values over each column in the node array is independent and consequently is embarassingly parallel. This code verson does not reflect the parallel computing as too many factors could detract from the intent of being didactic. Users could easily extend the loop withint the code to be parallel, however, this does require making additional copies of the node array which alone could be memory intensive. However, almost all test scoring applications in psychometrics are an embarassingly parallel problem as the test score for individual k does not depend on the scores for others, it depends only on the item parameters and their item responses, both of which are treated as fixed.

Example 3: A Simple ML Classifier with SGD

Imagine a group of test item developers would benefit from knowing whether the new items being developed are likely to have examinee response times that are fast or slow before those items have any field test and timing statistics. Such forecasting systems could guide item development plans and item writing workshops and be helpful in an array of test construction activities. Machine learning applications are premised on the idea that computers can be trained to learn from existing data and subsequently apply the results to form future classifications, or predictions, and such an application is suitable here.

For purposes of the example, assume response time data from a testing program exist for each test item and the data could be concatenated over years, over grades, over test forms, and over all examinees yielding an extremely large data file forming N total observations. In addition, assume all items in the set have a collection of observable features that are predictive of response time, such as readability indices, stem word length, content domain, cognitive complexity, item type and so on. Further, assume a binary outcome is associated with item response time such that $y_i = 1$ if the response to the ith item is fast and $y_i = 0$ otherwise and that the many item features can be used to form the model matrix $\boldsymbol{X}$ to serve as covariates in the regression. Response time is of course a real positive number, but here we bin into two categories to build an illustration that differs from the prior linear regression model using SGD.

The example described here yields an extremely large data file from which we intend to learn patterns and then make future predictions on other data not included in the training sample. Because the available data to be used for training is so large, logistic regression using something like iteratively reweighted least squares is not feasible. In this case, an alternative classifier can be constructed using SGD for training allowing for computation to occur in an efficient way. To show a contrast, in batch gradient descent with N observations, L parameters, and using K epochs,

the gradient would be evaluated $N \times K \times L$ times. The number for K is generally arbitrary, but is often selected as a very large value, such as $K = 1000$. Simply to be conceptual, imagine $N = 1,000,000$ and $L = 3$; then, batch gradient descent would evaluate 3 billion gradient terms!

Instead, SGD can reduce this computational burden and evaluate only $K \times L$ gradient terms to achieve comparable results. In fact, because SGD randomly samples from the full set of observations, it may be helpful to only randomly stream in portions of the full available data and sample from within those portions repeatedly. This could alleviate some overhead space needed to be reserved for computing. If mini-batch gradient descent were used, then $N_i \times K \times L$ gradient terms would be evaluated, where N_i represents a subset of N. The reduction in the number of gradient evaluations is impressive and makes very large problems feasible.

Here, let the objective function for the logistic model be $\mathcal{J}_i(\boldsymbol{\theta}) = -y_i \log(z_i) + (1 - y_i)(1 - z_i)$ and then

$$\nabla \mathcal{J}_i(\boldsymbol{\theta}) = (z_i - y_i)' \boldsymbol{X}_i \tag{33}$$

where the following familiar logistic (sigmoid) function is used

$$\Pr(y_i = 1|, \boldsymbol{X_i}, \boldsymbol{\beta}) = z_i = \frac{1}{1 + \exp(-(\boldsymbol{X}_i \boldsymbol{\beta}))}. \tag{34}$$

The gradient in (33) can be used in Equation (14) and the steps in Algorithm (1) would be implemented for optimization. A concept only noted in passing is that α is a learning rate and selecting values for this scalar are not fully explored here. Pragmatically speaking, choosing values for the learning rate parameter is a harder problem that it may seem at the surface and there is a good deal of research to further explore. A "large" value for learning is generally desired for initial evaluations of the gradient allowing the algorithm to take large steps towards the optimal value. However, using that same large value when approaching the optimum may cause for the algorithm to jump over the optimal value and not converge to the expected result. Hence, the learning rate "schedule" is a concept that adjusts the learning rate to smaller values as the algorithm proceeds. In addition, the example here assumes $L < N$, in which case standard regression models are used. In fact, many machine learning applications involve scenarios where $N < L$, in which case other approaches such as Ridge or Lasso regressions may prove useful, but can adopt the general SGD framework described in Algorithm (1).

Generalizations of this concept could be further developed to explore IRT forecasting problems where a sense of the future item parameters could be obtained from features associated with items or in training a learning management system to associate targeted instructional content to an examinee based on their test performance.

Discussion

The ideas here are but a subset of useful numerical methods. However, the subset represents a core set of methods that can be combined or generalized and broadly applied in psychometric application building. Hopefully, these recipes encourage even further study and conversations in methods that are necessary ingredients to support the ideas of computational psychometrics. While some specific recipes for computing are presented here, it's important not to lose sight of the primary motivation and aim of this work. Specifically, scalable psychometric solutions will be more

viable with better advancement towards efficient computational recipes and no longer conflating algebraic representations to also be the computational solutions. The aim of any computational psychometrician should be to exploit any special structures and find reduceable solutions to large problems.

This work is also intended to fit into a larger conversation of the future of psychometrics. The skill set for building scalable psychometric applications means innovators need foundational numerical approaches, but also skills in software development to prototype, finalize, and deploy at scale applications to support those ideas. Commercial cloud-based infrastructures are more readily available and provide platforms to take advantage of distributed computing. Nonetheless, the first step should be to build an efficient method and then secondly scale that method with computational advantages offered.

Appendix A: Computational Formulae

Proof for Reducing $(\boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D} + \boldsymbol{I})^{-1}$

Let $\boldsymbol{Z}'\boldsymbol{Z} = \boldsymbol{Q}_1\boldsymbol{\lambda}_1\boldsymbol{Q}_1'$ and $\boldsymbol{D} = \boldsymbol{Q}_2\boldsymbol{\lambda}_2\boldsymbol{Q}_2' = \boldsymbol{\lambda}_2$ given that $\boldsymbol{D} = d\boldsymbol{I}$, then

$$\begin{aligned}\boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D} + \boldsymbol{I} &= (\boldsymbol{Q}_1\boldsymbol{\lambda}_1\boldsymbol{Q}_1')(\boldsymbol{Q}_2\boldsymbol{\lambda}_2\boldsymbol{Q}_2') + \boldsymbol{I} &(35)\\ &= (\boldsymbol{Q}_1\boldsymbol{\lambda}_1\boldsymbol{Q}_1')\boldsymbol{\lambda}_2 + \boldsymbol{I} &(36)\end{aligned}$$

The commutative property for matrices holds if and only if $d\boldsymbol{I}$, hence $\boldsymbol{Q}_1'\boldsymbol{\lambda}_2 = \boldsymbol{\lambda}_2\boldsymbol{Q}_1'$. Then, $(\boldsymbol{Q}_1\boldsymbol{\lambda}_1\boldsymbol{Q}_1')\boldsymbol{\lambda}_2 + \boldsymbol{I} = (\boldsymbol{Q}_1\boldsymbol{\lambda}_1\boldsymbol{\lambda}_2\boldsymbol{Q}_1') + \boldsymbol{I}$. Finally, we can take advantage of the identity matrix and co-diagonalize its elements alongside the eigenvalues $\boldsymbol{\lambda}_1\boldsymbol{\lambda}_2 = \boldsymbol{\lambda}^* = diag\{\lambda_{11}d + 1, \lambda_{12}d + 1, \ldots, \lambda_{n1}d + 1\}$. Assembling all components then gives

$$\boldsymbol{Q}_1\boldsymbol{\lambda}_1\boldsymbol{\lambda}_2\boldsymbol{Q}_1' + \boldsymbol{I} = \boldsymbol{Q}_1(\boldsymbol{\lambda}_1\boldsymbol{\lambda}_2 + \boldsymbol{I})\boldsymbol{Q}_1' = \boldsymbol{Q}_1\boldsymbol{\lambda}^*\boldsymbol{Q}_1' \tag{37}$$

Finally, the inverse of the sums is simplified to

$$(\boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D} + \boldsymbol{I})^{-1} = \boldsymbol{Q}_1\boldsymbol{\lambda}^{*-1}\boldsymbol{Q}_1'. \tag{38}$$

Equation (38) is useful in iterative algorithms as it precomputes an expensive component, $\boldsymbol{Q}_1$, initially and then it is repeatedly reused in subsequent iterations. Because updating $\boldsymbol{\lambda}^*$ is simple and it is diagonal, it permits a trivial inverse. Collectively, the special structures in this instance are exploited to reduce a larger computational burden into one that is much less expensive within iterative algorithms.

General Remark on the Proof

It is important to note that the preceding proof applies under conditions when $\boldsymbol{D} = d\boldsymbol{I}$, in which case the commutative property holds between $\boldsymbol{D}$ and some other (conformable) matrix. However, when that condition is not true, we arrive at a slightly different problem. Suppose $\boldsymbol{D}$ is diagonal, but the elements along the diagonal are not constant. Then, we may write $(\boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D} + \boldsymbol{I})^{-1} = (\boldsymbol{Q}\boldsymbol{\lambda}\boldsymbol{Q}' + \boldsymbol{I})^{-1} = \boldsymbol{Q}(\boldsymbol{\lambda} + \boldsymbol{I})^{-1}\boldsymbol{Q}'$ where now $\boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D} = \boldsymbol{Q}\boldsymbol{\lambda}\boldsymbol{Q}'$.

This is interesting, but not entirely helpful within an iterative algorithm as $\boldsymbol{D}$ cannot be factored out of the decomposition. Hence, it requires computing $\boldsymbol{Z}'\boldsymbol{Z}\boldsymbol{D}$ at each iteration even though only $\boldsymbol{D}$ is changing. At the current time, there is no accepted method for the inverse of matrix sums under this condition. There are some concepts that may later be considered using eigenvalue perturbations. That is, if we can first compute the decomposition $\boldsymbol{Z}'\boldsymbol{Z}$, then update this with the changing values of $\boldsymbol{D}$ to yield an approximation of $\boldsymbol{Q}\boldsymbol{\lambda}\boldsymbol{Q}'$, then we might be able to efficiently find the sum of the inverse when $\boldsymbol{D}$ is diagonal, but not with constant elements. This remains a numerical challenge to explore.

Likelihood Function Details

Each individual domain likelihood, $\mathcal{L}(\theta_P; \boldsymbol{z}_p)$, is composed of a mixture of binary and polytomous test items with known item parameters. Let z_{ijp} denote the observed response of the ith examinee to the jth item in the pth dimension, then

$$\mathcal{L}(\theta_P; \boldsymbol{z}_p) = \mathcal{L}_1(\theta_P; \boldsymbol{z}_p)\mathcal{L}_2(\theta_P; \boldsymbol{z}_p)$$

$$\mathcal{L}_1(\theta_P; \boldsymbol{z}_p) = \prod_{j \in p} \left[c_j + \frac{1 - c_j}{1 + \exp[-Da_j(\theta_P - b_j)]} \right]^{z_{ijp}} \left[1 - \left(c_j + \frac{1 - c_j}{1 + \exp[-Da_j(\theta_P - b_j)]} \right) \right]^{1 - z_{ijp}}$$

where $j \in p$ is used to mean there is a collection of j items uniquely associated with dimension p, c_j is the lower asymptote of the item response curve (i.e., the guessing parameter), a_j is the slope of the item response curve (i.e., the discrimination parameter), $b_j$is the location parameter, and D is a constant, by default fixed at 1.7. Then the items scored in multiple categories takes the form of the graded response model as

$$\mathcal{L}_2(\theta_P; \boldsymbol{z}_p) = \prod_{j \in p} \Pr(z_{ijp} | \theta_P)$$

$$\Pr(z_{ijp}|\theta) = \begin{cases} \frac{1}{1+e^{Da_j(\theta_P - b_{j1})}}, z_{ijp} = 0 \\ \frac{1}{1+e^{Da_j(\theta_P - b_{j,z+1})}} - \frac{1}{1+e^{Da_j(\theta_P - b_{jz})}}, \ 0 < z_{ijp} < K \\ \frac{1}{1+e^{-Da_j(\theta_P - b_{jK})}}, \ z_{ijp} = K \end{cases} \tag{39}$$

where b_K denotes the kth step and the other definitions used for the binary model apply here.

Appendix B: Computational Code Examples

Code Examples For Linear Systems

```
### Simple implementation of error in variables linear model
eivreg <- function(y,X,r){
  vars <- apply(X, 2, var)
  N <- length(y)
  S <- diag((1-r)*vars * N)
  XtX.S <- crossprod(X) - S
  L <- chol(XtX.S)
  yprime <- crossprod(X,y)
  B <- backsolve(L,forwardsolve(t(L),yprime))
  B
}

N <- 5000
t1 <- rnorm(N)
t2 <- rnorm(N)
x1 <- (t1 + rnorm(N, sd = .4))
x2 <- (t2 + rnorm(N, sd = .3))
y <- 1 + 2*t1 + .5*t2 + rnorm(N)
X <- model.matrix(y ~ x1 + x2)
r <- c(1, var(t1)/var(x1), var(t2)/var(x2))
eivreg(y,X,r)

### R Code for least squares using decomposition
X <- matrix(c(1,1,1,1,10,3,15,4), ncol=2)
y <- c(5,10,15,20)
yprime <- crossprod(X,y)
L <- chol(crossprod(X))
### Step 1 solve for Z
Z <- forwardsolve(t(L),yprime)
### Step 2 solve for B. These are the least squares estimates
(B <- backsolve(L,Z))
lm(y~X[,2]) # Built in function for a check

### R Code for weighted least squares using decomposition and
   elementwise calculation
X <- matrix(c(1,1,1,1,10,3,15,4), ncol=2)
y <- c(5,10,15,20)
V <- diag(runif(4)) #simulate weights on diagonal
X2 <- 1/diag(V) * X
yprime <- crossprod(X2,y)
L <- chol(crossprod(X,X2))
### Step 1 solve for Z
Z <- forwardsolve(t(L),yprime)
### Step 2 solve for B. These are the WLS estimates
```

```
(B <- backsolve(L,Z))
```

Code Examples For EAP Examples

```
library(mvtnorm)
library(statmod)

grm <- function (theta, d, score, a, D = D) {
  maxD <- length(d)
  if(score == 0) {
    pr <- 1/(1 + exp(D*a*(theta - d[(score+1)])))
  }
  else if(score == maxD) {
    pr <- 1/(1 + exp(-D*a*(theta - d[score])))
  } else {
    pr <- 1/(1 + exp(D*a*(theta - d[(score+1)])))
      - 1/(1 + exp(D*a*(theta - d[score])))
  }
  pr
}

paramList.mirt <- function(dat, dim.order = NULL){
  tmp <- split(dat, dat$strand)
  tmp <- tmp[dim.order]
  nms <- names(tmp)
  num.dimensions <- length(tmp)
  mat <- vector("list", num.dimensions)
  for(i in 1:num.dimensions){
    vars <- c('a', paste('b', 1:5, sep=''))
    poly <- subset(tmp[[i]], model == 'GRL')
    poly <- poly[, vars]
    a_vector <- as.numeric(poly$a)
    poly <- as.data.frame(t(poly[, vars[-1]]))
    py <- as.list(poly)
    py <- lapply(py, function(x) x[!is.na(x)])

    ### MC Items
    mc <- subset(tmp[[i]], model == '3pl')
    mat[[i]] <- list("3pl" = list(a = mc$a , b = mc$b, c = mc$c),
    gpcm = list(a = a_vector, d = py))
  }
  names(mat) <- dim.order
  mat
}

gauss.proc <- function(Q, num.dimensions = 4, mu, sigma, method = c(
   'gh', 'ghc', 'es', 'ghr')){
```

```
  sigma <- as.matrix(sigma)
  if (method == 'gh'){
    idx <- as.matrix(expand.grid(rep(list(1:Q),num.dimensions)))
    gh <- gauss.quad(Q, kind='hermite')
    nodes <- gh$nodes * sqrt(2)
    whts <- gh$weights * pi^-num.dimensions/2
    nodes <- matrix(nodes[as.vector(idx)], ncol = num.dimensions)
    whts <- matrix(whts[as.vector(idx)], ncol = num.dimensions)
    whts <- whts_orig <- exp(rowSums(log(whts)))
    whts <- (dmvnorm(nodes, mean = mu, sigma = sigma)/dmvnorm(nodes,
        mean = mu, sigma = diag(1,num.dimensions))) * whts
    rotate <- FALSE
  } else if (method == 'ghc'){
    idx <- as.matrix(expand.grid(rep(list(1:Q),num.dimensions)))
    gh <- gauss.quad(Q, kind='hermite')
    nodes <- gh$nodes  * sqrt(2)
    whts <- gh$weights * pi^-num.dimensions/2
    nodes <- matrix(nodes[as.vector(idx)], ncol = num.dimensions)
       %*% chol(sigma)
    whts <- matrix(whts[as.vector(idx)], ncol = num.dimensions)
    whts <- whts_orig <- exp(rowSums(log(whts)))
    rotate <- TRUE
  } else if (method == 'ghr') {
    nodes <- rmvnorm(Q, mean = mu, sigma = sigma)
    whts <- whts_orig <- 1
    idx <- NULL
    rotate <- TRUE
  } else {
    idx <- as.matrix(expand.grid(rep(list(1:Q),num.dimensions)))
    nodes <- seq(-4, 4, length.out = Q)
    nodes <- matrix(nodes[as.vector(idx)], ncol = num.dimensions)
    whts <- whts_orig <- dmvnorm(nodes, mean = mu, sigma)
    rotate <- FALSE
  }
  list(nodes = nodes, whts = whts, idx = idx, Q = Q, rotate = rotate
     , whts_orig=whts_orig)
}

mirt.eap <- function(x, params, mu, sigma, D = 1.7, dim.order = NULL
   , skip = FALSE, grid, refine.steps = 1){

  Q <- grid$Q
  rotate <- grid$rotate
  num.dimensions <- length(mu)
  sigma <- as.matrix(sigma)

  x <- split(x, x$strand)
```

```
scoreMat <- vector("list", num.dimensions)
names(scoreMat) <- names(x)[1:num.dimensions]
for(i in 1:length(x)){
  scoreMat[[i]] <- x[[i]][, c('score', 'model')]
}

### Must reorder so things are in same order as covariance matrix
scoreMat <- scoreMat[dim.order]

### Do error checks here
if(skip == FALSE){
  if(length(mu) != ncol(sigma)) stop('The mean vector and
     covariance matrix are not conformable')
  if(any(names(rr) != names(scoreMat))) stop('Your scoreMat is
     ordered as ', names(scoreMat),
    ' but your item parameters are ordered as ', names(params))
  if(Q > 50) stop('You are using a LOT of quadrature points. If
     you are sure you want to do this, use skip = TRUE to bypass
     this message')}

nodes <- grid$nodes
whts <- grid$whts

for(l in 1:refine.steps){ ## loop for centering

if(l > 1) {
  SE <- sqrt(diag(vcov))
  nodes <- sweep(sweep(nodes, 2, SE, FUN='*'), 2, result, FUN='+')
  whts <- (dmvnorm(nodes, mean = mu, sigma = sigma)/dmvnorm(nodes,
      mean = rep(0,4), sigma = diag(.8,4))) * grid$whts_orig
}

mat <- vector("list", num.dimensions)
for(k in 1:num.dimensions){
  a <- params[[k]]$'3pl'$a
  b <- params[[k]]$'3pl'$b
  c <- params[[k]]$'3pl'$c
  aa <- params[[k]]$gpcm$a
  d1 <- params[[k]]$gpcm$d
  x1 <- scoreMat[[k]]$score[which(scoreMat[[k]]$model == '3pl')]
  x2 <- scoreMat[[k]]$score[which(scoreMat[[k]]$model != '3pl')]
  if(rotate) {
    gh <- nodes[,k]
    Q <- length(gh)
  } else {
    gh <- unique(nodes[,k])
  }
```

```
    if(length(x1) > 0){
      tmp <- sapply(1:Q, function(i) c + (1 - c) / (1 + exp(-D * a *
          (gh[i] - b))))
      res1 <- sapply(1:Q, function(i) prod(dbinom(x1, 1, tmp[,i])))
      if(rotate){
        t1 <- res1
      } else {
        t1 <- res1[as.vector(grid$idx[,k])]
      }
    } else t1 <- NULL
    if(length(x2) > 0){
      res2 <- sapply(1:Q, function(i) prod((mapply(grm, d1, aa,
         theta = gh[i], score = x2, D = D))))
      if(rotate){
        t2 <- res2
      } else {
        t2 <- res2[as.vector(grid$idx[,k])]
      }
    } else t2 <- NULL
    mat[[k]] <- cbind(t1, t2)
  }

  ### rn is for numerator and is likelihood * prior
  rn <- apply(cbind(do.call(cbind, mat), whts),1,prod)
  ### rd is 'result denominator' and is normalizing constant
  rd <- sum(rn)
  result <- colSums(nodes * rn)/rd
  ## full covariance matrix
  vcov <- sapply(1:num.dimensions, function(j) sapply(1:num.
     dimensions, function(i)
    sum((nodes[,i] - result[i]) * (nodes[,j] - result[j]) * rn)/rd))


  } ### end centering loop
  structure(list(coefficients = result, vcov = vcov, Dimensions =
     dim.order), class = 'mirt.eap')
}
```

Code to Compute Multivariate as Product of Univariate Densities

```
library(mvtnorm)
sigma <- matrix(c(1,.5,.5,1),2,2)
L <- chol(sigma)
dmvnorm(c(1,-1), mean = c(0,0), sigma = sigma)
prod(1/diag(L)) * prod(dnorm(forwardsolve(t(L), c(1, -1))))
```

References

Ackerberg, D. (2009). A new use of importance sampling to reduce computational burden in simulation estimation. *Quant Mark Econ*, *7*, 343-376.

ACTNext. (2019). *Computational psychometrics: A field guide* (Tech. Rep.). https://actnext.org/research-and-projects/computational-psychometrics-field-guide/: ACT.

Allen, N., Donoghue, J., & Schoeps, T. (2001). *The NAEP 1998 technical report* (Tech. Rep.). https://nces.ed.gov/nationsreportcard/pubs/main1998/2001509.asp: U.S. Department of Education. Office of Educational Research and Improvement. National Center for Education Statistics.

Antal, T., & Oranje, A. (2007). *Adaptive numerical integration for item response theory* (Tech. Rep.). https://files.eric.ed.gov/fulltext/EJ1111562.pdf: ETS.

Bates, D. (2004a, June). Least squares calculations in R. *R News*, *4*(1), 17–20.

Bates, D. (2004b). *Sparse matrix representations of linear mixed models* (Tech. Rep.). http://www.stat.wisc.edu/ bates/reports/MixedEffects.pdf: University of Wisconsin-Madison.

Bock, R. D., & Mislevy, R. J. (1982). Adaptive EAP estimation of ability in a microcomputer environment. *Applied Psychological Measurement*, *6*(4), 431-444.

Cai, L., Yang, J., & Hansen, M. (2011). Generalized full-information item bifactor analysis. *Psychological methods*, *16*(3), 221-248.

Chalmers, R. (2012, 04). Mirt: A multidimensional item response theory package for the R environment. *JSS Journal of Statistical Software*, *48*.

Chowdhary, K., Salloum, M., Debusschere, B., & Larson, V. E. (2015). Quadrature methods for the calculation of subgrid microphysics moments. *Monthly Weather Review*, *143*(7), 2955-2972.

Cizek, P., & Cizkova, L. (2004). Iterative methods for solving linear systems. In J. Gentle, W. Härdle, & Y. Mori (Eds.), *Handbook of compuational statistics* (p. 120-126). New York: Springer.

Davis, T., & Hager, W. (2005, 01). Row modifications of a sparse cholesky factorization. *SIAM J. Matrix Analysis Applications*, *26*, 621-639.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, *39*, 1-38.

Doran, H. (2014). Methods for incorporating measurement error in value-added models and teacher classifications. *Statistics and Public Policy*, *1*(1), 114-119. Retrieved from https://doi.org/10.1080/2330443X.2014.955228

Doran, H., Bailey, P., Buehler, E., & joo Lee, S. (2021). Dire: Linear regressions with a latent outcome variable [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=Dire (R package version 1.0.3)

Elvira, V., Martino, L., & Closas, P. (2021). Importance gaussian quadrature. *IEEE Transactions on Signal Processing*, *69*, 474-488. Retrieved from http://dx.doi.org/10.1109/TSP.2020.3045526

Ferrando, P., & Lorenzo-Seva, U. (2016). A note on improving EAP trait estimation in oblique factor-analytic and item response theory models. *Psicologica*, *37*(2), 235-247.

Fletcher, R. (1987). *Practical methods of optimization* (Second ed.). New York, NY, USA: John Wiley & Sons.

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2004). *Bayesian data analysis* (2nd ed. ed.). Chapman and Hall/CRC.

Genz, A., & Kass, R. E. (1997). Subregion-adaptive integration of functions having a dominant peak. *Journal of Computational and Graphical Statistics*, *6*(1), 92-111.

Hsu, Y., Ackerman, T. A., & Fan, M. (1999). The relationship between the bock-aitkin procedure and the em algorithm for irt model estimation..

Jäckel, P. (2005). *A note on multivariate gauss-hermite quadrature.* (Tech. Rep.). http://www.jaeckel.org/ANoteOnMultivariateGaussHermiteQuadrature.pdf.

Judd, K. L., Maliar, L., & Maliar, S. (2011). Numerically stable and accurate stochastic simulation approaches for solving dynamic economic models. *Quantitative Economics*, *2*(2), 173-210. Retrieved from https://onlinelibrary.wiley.com/doi/abs/10.3982/QE14

Kessy, A., Lewin, A., & Strimmer, K. (2018). Optimal whitening and decorrelation. *The American Statistician*, *72*(4), 309-314. Retrieved from https://doi.org/10.1080/00031305.2016.1277159

King, G. (1998). *Unifying political methodology: The likelihood theory of statistical inference*. Ann Arbor: University of Michigan Press.

Laird, N. M., & Ware, J. H. (1982). Random-effects models for longitudinal data. *Biometrics*, *38*(4), 963–974. Retrieved from http://www.jstor.org/stable/2529876

Lee, S. X., Leemaqz, K. L., & McLachlan, G. J. (2016). A simple parallel EM algorithm for statistical learning via mixture models. In *2016 international conference on digital image computing: Techniques and applications (dicta)* (p. 1-8).

Lira, M., Iyer, R., Trindade, A., & Howle, V. (2016). QR versus cholesky: A probabilistic analysis. *International Journal of Numerical Analysis and Modeling*, *13*(1), 114–121. (Publisher Copyright: © 2016 Institute for Scientific Computing and Information.)

Liu, Q., & Pierce, D. A. (1994). A note on gauss-hermite quadrature. *Biometrika*, *81*(3), 624–629. Retrieved from http://www.jstor.org/stable/2337136

Lockwood, J. R., & McCaffrey, D. F. (2020). Recommendations about estimating errors-in-variables regression in stata. *The Stata Journal*, *20*(1), 116-130. Retrieved from https://doi.org/10.1177/1536867X20909692

McCulloch, C. E., & Searle, S. R. (2001). *Generalized, linear, and mixed models*. New York: John Wiley and Sons.

McLean, R. A., Sanders, W. L., & Stroup, W. W. (1991). A unified approach to mixed linear models. *The American Statistician*, *45*(1), pp. 54-64. Retrieved from http://www.jstor.org/stable/2685241

Mislevy, R. J. (1984). Estimating latent distributions. *Psychometrika*, *49*(3), 359-381.

Nab, L., van Smeden, M., Keogh, R. H., & Groenwold, R. H. (2021). Mecor: An R package for measurement error correction in linear regression models with a continuous outcome. *Computer Methods and Programs in Biomedicine*, *208*, 106238.

Pinheiro, J., & Bates, D. (1994, 09). Approximations to the log-likelihood function in the non-linear mixed-effects model. *Journal of Computational and Graphical Statistics*, *4*. doi: 10.1080/10618600.1995.10474663

Quarteroni, A., Saleri, F., & Gervasio, P. (2010). *Scientific computing with matlab and octave. 3rd ed*. Springer.

Rosseel, Y. (2021). Evaluating the observed log-likelihood function in two-level structural equation modeling with missing data: From formulas to R code. *Psych*, *3*(2), 197–232. Retrieved from https://www.mdpi.com/2624-8611/3/2/17

SAS Documentation 14.2. (n.d.). Estimating fixed and random effects in the mixed model [Computer software manual]. (Online Help Manual)

Searle, S. (1982). *Matrix algebra useful for statistics*. New York: John Wiley and Sons.

Shamir, O., & Zhang, T. (2013, 17–19 Jun). Stochastic gradient descent for non-smooth

optimization: Convergence results and optimal averaging schemes. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (Vol. 28, pp. 71–79). Atlanta, Georgia, USA: PMLR. Retrieved from https://proceedings.mlr.press/v28/shamir13.html

StataCorp. (n.d.). Errors-in-variables regression [Computer software manual]. Retrieved from https://www.stata.com/manuals/reivreg.pdf (Online Help Manual)

Stringer, A. (2021). *Implementing approximate bayesian inference using adaptive quadrature: the aghq package.*

Tuerlinckx, F., Rijmen, F., Verbeke, G., & De Boeck, P. (2006, 12). Statistical inference in generalized linear mixed models: A review. *The British journal of mathematical and statistical psychology*, *59*, 225-55.

von Davier, A. A., Deonovic, B., Yudelson, M., Polyak, S. T., & Woo, A. (2019). Computational psychometrics approach to holistic learning and assessment systems. *Frontiers in Education*, *4*, 69. Retrieved from https://www.frontiersin.org/article/10.3389/feduc.2019.00069

Woodbury, M. A. (1950). Inverting Modified Matrices. In J. Kuntzmann (Ed.), . Princeton, NJ: Princeton University.

Zhang, F. (1999). *Matrix theory: Basic results and techniques*. New York: Springer.