Behavior Research Methods (2023) 55:4403–4418
https://doi.org/10.3758/s13428-022-02024-x

Accelerating item factor analysis on GPU with `Python` package `xifa`

Po-Hsien Huang[1]

Accepted: 8 November 2022 / Published online: 10 January 2023

Abstract
Item parameter estimation is a crucial step when conducting item factor analysis (IFA). From the view of frequentist estimation, marginal maximum likelihood (MML) seems to be the gold standard. However, fitting a high-dimensional IFA model by MML is still a challenging task. The current study demonstrates that with the help of a GPU (graphics processing unit) and carefully designed vectorization, the computational time of MML could be largely reduced for large-scale IFA applications. In particular, a `Python` package called `xifa` (accelerated item factor analysis) is developed, which implements a vectorized Metropolis–Hastings Robbins–Monro (VMHRM) algorithm. Our numerical experiments show that the VMHRM on a GPU may run 33 times faster than its CPU version. When the number of factors is at least five, VMHRM (on GPU) is much faster than the Bock–Aitkin expectation maximization, MHRM implemented by `mirt` (on CPU), and the importance-weighted autoencoder (on GPU). The GPU-implemented VMHRM is most appropriate for high-dimensional IFA with large data sets. We believe that GPU computing will play a central role in large-scale psychometric modeling in the near future.

Keywords Item factor analysis · Item response theory · Deep learning · Parallel computing

Item factor analysis (IFA; Bock, Gibbons, & Muraki, 1988) is a statistical technique that aims to explain the dependency among item-level data by introducing latent factors. Usually, the item-level data are binary or, more generally, ordered categorical. Ability tests with yes–no questions provide a simple example. A generalization of this is the Likert-type response scale used in attitude and personality questionnaires (*strongly disagree*, *disagree*, *neither*, *agree*, and *strongly agree*). If modeling category response probability is the main purpose, IFA can be also regarded as a methodology of item response theory (IRT, for an equivalence, see Takane & de Leeuw, 1987). In practice, IFA is often used to evaluate the underlying factor structure of a psychological measurement. This structure provides a basis for latent trait estimation and dimension reduction, which are useful in statistical tasks such as regression, classification, and clustering.

✉ Po-Hsien Huang
psyphh@nccu.edu.tw

[1] Department of Psychology, National Chengchi University, 64, Section 2, Zhi-Nan Road, Taipei City, Taiwan

Item parameter estimation is a crucial step in IFA, which can be achieved by frequentist or Bayesian estimation (see Chen, Li, Liu, & Ying, 2021; Wirth & Edwards, 2007, for reviews). From the view of frequentist estimation, *marginal maximum likelihood* (MML; Bock & Lieberman, 1970) seems to be the gold standard because of its consistency, and asymptotical efficiency and normality. MML tries to find an estimate that maximizes the so-called marginal likelihood. To determine marginal likelihood in the optimization process, MML must integrate over M latent factors. If the number of latent factors is small (e.g., $M < 5$), the integral can be efficiently computed by Gauss–Hermite quadrature (e.g., Bock & Aitkin, 1981; Bock & Lieberman, 1970; Gibbons & Hedeker, 1992) or by its adaptive variation (e.g., Schilling & Bock, 2005). Otherwise, stochastic algorithms are used. Famous stochastic algorithms for IFA include Monte Carlo expectation maximization (MCEM; e.g., Meng & Schilling, 1996; Song & Lee, 2005), stochastic expectation maximization (StEM; including an improved version, see Zhang, Chen, & Liu, 2020), and Metropolis–Hastings Robbins–Monro (MHRM; e.g., Cai, 2010a, b) algorithms. Some of these implementations are available in mainstream IFA programs such as `IRTPRO` (Cai, Du Toit, & Thissen, 2011), `flexMIRT` (Cai, 2017),

`Mplus` (Muthén & Muthén, 1998–2017), and `mirt` (Chalmers, 2012).

Nevertheless, fitting a high-dimensional IFA model by MML is still a time-consuming task. For example, MML might take about 36 min for fitting an IFA model with ten factors to a data set with 2500 observations and 300 items (see Chen, Li, & Zhang, 2019). Note that 36 min is just for single analysis. In practice, researchers may try many IFA models under different data conditions for searching an optimal result, which definitely spends more time. The computational burden of MML precludes researchers from exploring data-model fit thoroughly. Even for a relatively small IFA model, it still becomes very time-consuming when implementing resampling methods for robust statistical inferences (e.g., Liou & Yu, 1991; Patton, Cheng, Yuan, & Diao, 2014).

To make up for the deficiency of MML, psychometricians develop less computationally intensive procedures, including variational inference (VI) methods (Cho, Wang, Zhang, & Xu, 2020; Hui, Warton, Ormerod, Haapaniemi, & Taskinen, 2017; Wu, Davis, Domingue, Piech, & Goodman, 2020), the constrained joint maximum likelihood (CJML) (Chen et al., 2019; Chen, Li, & Zhang, 2020), and the importance-weighted autoencoder (IWAE) method (Urban & Bauer, 2021). However, these alternative methods are still restrictive in several ways. (1) The consistency of CJML and VI require a large number of items for each latent trait (Chen et al., 2019; Cho et al., 2020). Most empirical settings, however, use less than six items for measuring a latent trait (see Jackson, Gillaspy Jr, & Purc-Stephenson, 2009, for a review). (2) Both the theoretical properties and the empirical performance of CJML and IWAE rely on tuning parameters. The tuning parameter values are usually determined by cross-validation, which complicates the implementation of these methods. (3) The existing findings on statistical inferences for MML (for reviews, see Swaminathan, Hambleton, & Rogers, 2006; Yuan, Cheng, & Patton, 2014) cannot be directly applied to the above alternative methods. Based on these reasons, we believe that most IFA users still prefer using MML unless its implementation is totally infeasible.

If we stay with the implementations of MML, we can still search for computationally more feasible answers. The aim of the current study is to invoke help from the GPU (graphics processing unit) and a carefully designed vectorization to handle large-scale IFA applications. The GPU is an electronic circuit that helps the CPU (central processing units) for handling computer graphics. After the release of CUDA (NVIDIA, Vingelmann, & Fitzek, 2020), the GPU gradually entered scientific computing due to its parallelization capability (Keckler, Dally, Khailany, Garland, & Glasco, 2011; Nickolls & Dally, 2010).

We believe that parallelization with GPU has two advantages. First, GPU machines are readily available today. Some online coding platforms such as Colab and Kaggle even provide free GPU computing resources. In contrast, powerful multi-core CPU workstations are difficult to access for end users. Second, it is not necessary to write source code for GPU computing. Today, GPU is supported by many user-friendly deep learning libraries such as `TensorFlow` (Abadi et al., 2015), `PyTorch` (Paszke et al., 2019), and `Jax` (Bradbury et al., 2018). These libraries provide highly optimized functions with both CPU and GPU backends depending on the availability of GPU. Therefore, programmers are only required to vectorize most operations by using these functions.

Note that accelerating psychometric modeling by parallelization is not new. There were several works considering this issue on either GPU or CPU (e.g., von Davier, 2017; Loossens et al., 2021; Sheng, Welling, & Zhu, 2014, 2015; Verdonck, Meers, & Tuerlinckx, 2016). However, none of them study GPU computing for high-dimensional IFA. The unique contribution of the present work is to introduce a `Python` package called `xifa` (accelerated item factor analysis), which implements a vectorized MHRM (VMHRM) algorithm for a wide class of high-dimensional IFA models. The vectorized algorithm could be greatly speeded up on GPU. In addition, the present work establishes benchmarks for `xifa` by empirically comparing it with some popular or state-of-the-art implementations, including Bock–Aitkin EM (BAEM) (Bock & Aitkin, 1981), MHRM, and IWAE. As we shall see in our simulations, the VMHRM on GPU could run 33 times faster than its CPU version. We believe this progress is a breakthrough.

The article is organized as follows: First, an IFA framework and the steps of MHRM are presented. Second, we introduce the VMHRM and demonstrate how to use `xifa`. The technical details of VMHRM can be found in Appendix A and B. Third, numerical experiments of algorithm comparison are executed. Fourth, a real data example illustrates the powerfulness of our approach. Finally, merits and limitations of the current study are discussed.

Item factor analysis and marginal likelihood

A framework for item factor analysis

Let $v = (v_1, v_2, ..., v_I)$ denote an I-dimensional response vector of I polytomous items. For each item v_i, C_i denotes the number of response categories, that is, v_i takes on a value in $\{0, 1, ..., C_i - 1\}$. The value of v_i is regarded to satisfy an ordinal scale (Stevens, 1946). To make things a bit simpler, we assume $C_i = C$ for now, and return to the general case later. The IFA characterizes the response probability of v_i as a function of an M-dimensional latent

factor vector, say $\eta = (\eta_1, \eta_2, ..., \eta_M)$. This probability is expressed as:

$$\pi_i(\eta) = (\pi_{i0}(\eta), \pi_{i1}(\eta), ..., \pi_{i(C-1)}(\eta)), \tag{1}$$

where $\pi_{ic}(\eta)$ is the conditional probability of the event $v_i = c$ given the trait level η, i.e., $\pi_{ic}(\eta) = \Pr(v_i = c|\eta)$. Note that η is a latent variable that cannot be directly observed. The latent factor is often assumed to be normally distributed with zero mean, i.e., $\Pr(\eta) = \text{Normal}(0, \Phi)$. In particular, the covariance matrix Φ is set to be standardized with $\phi_{mm'}$ being the correlation of η_m and $\eta_{m'}$.

The exact functional form of $\pi_i(\eta)$ depends on the IFA model class. For example, the graded response model (GRM; Samejima, 1969) assumes that:

$$\pi_{ic}(\eta) = \frac{1}{1+\exp\left(-\nu_{ic} - \lambda_i^T \eta\right)} - \frac{1}{1+\exp\left(-\nu_{i(c+1)} - \lambda_i^T \eta\right)}, \tag{2}$$

where ν_{ic} is the intercept for the c^{th} response category of item i such that $-\infty = \nu_{iC} < \nu_{i(C-1)} < ... < \nu_{i0} = \infty$, and $\lambda_i = (\lambda_{i1}, \lambda_{i2}, ..., \lambda_{iM})$ is an M-dimensional loading vector for item i. Another famous example is the generalized partial credit model (GPCM; Muraki, 1992) that assumes:

$$\pi_{ic}(\eta) = \frac{\exp\sum_{j=0}^{c}\left(\nu_{ij} + \lambda_i^T \eta\right)}{\sum_{k=0}^{C-1}\exp\sum_{j=0}^{k}\left(\nu_{ij} + \lambda_i^T \eta\right)}, \tag{3}$$

where $\nu_{i0} + \lambda_i^T \eta$ is defined as zero. Both GRM and GPCM use linear predictors of the form:

$$\tau_{ic} = \nu_{ic} + \lambda_i^T \eta. \tag{4}$$

Hence, GRM and GPCM can be formulated as generalized linear multivariate models (Fahrmeir & Tutz, 1994) if η can be directly observed. For more IFA model classes, refer to van der Linden (2016).

Marginal likelihood via MHRM

Consider a random sample $V = (v_n)_{n=1}^N$ of size N. Let θ denote the parameter vector containing all freely estimated model parameters including the intercepts (ν_{ic}), the loadings (λ_{im}), and the correlations among factors ($\phi_{mm'}$). To estimate θ, MML adopts the following log-marginal likelihood function:

$$\ell(\theta; V) = \frac{1}{N}\sum_{n=1}^{N}\ell(\theta; v_n), \tag{5}$$

where

$$\ell(\theta; v) = \log\left[\int \Pr(v|\eta;\theta)\Pr(\eta;\theta)\mathrm{d}\eta\right]. \tag{6}$$

Any maximizer $\widehat{\theta}$ for $\ell(\theta; V)$ is called an MML estimate of θ. When $M \geq 5$, the integral is generally evaluated by Monte Carlo methods, among which MHRM is one of the most often used.

Let $\mathrm{H} = (\eta_n)_{n=1}^N$ denote the $N \times M$ array with η_n as the true factor level corresponding to v_n. The MHRM can be understood as an expectation maximization (EM; Dempster, Laird, & Rubin, 1977) algorithm that augments latent factors into the so-called complete data likelihood, denoted by $\ell(\theta; V, \mathrm{H})$, and then maximizes this likelihood to obtain an MML estimate. In particular, MHRM uses the Metropolis–Hastings (MH) method to sample latent factors from their posterior distributions and then implements a Robbins–Monro (RM) step to update the current parameter estimate. An implementation of MHRM is presented in Algorithm 1 (for details, see Cai, 2010a).

Initialization: Initialize $\widehat{\theta}$ (initial estimate), $\{\widetilde{\eta}_n^{(0)}\}_{n=1}^N$ (initial latent scores), σ^2_{jump} (MH jumping variance), s (learning rate), J (length of chain), K (MH samples), and $t = 1$.
while $\widehat{\theta}$ *changes significantly* **do**
  Metropolis-Hastings Sampling:
  for $n = 1, 2, ..., N$ **do**
    for $j = 1, 2, ..., J$ **do**
      Generate candidate: $\widetilde{\eta}'_n = \widetilde{\eta}_n^{(j-1)} + \widetilde{\zeta}_n^{(j)}$ with $\widetilde{\zeta}_n^{(j)} \sim \text{Normal}(0, \sigma^2_{\text{jump}} I_M)$;
      Compute acceptance probability: $\alpha_n = \min\left\{\frac{\ell(\widehat{\theta}; v_n, \widetilde{\eta}'_n)}{\ell(\widehat{\theta}; v_n, \widetilde{\eta}_n^{(j-1)})}, 1\right\}$, where $\ell(\theta; v, \eta) = \log\Pr(v|\eta;\theta) + \log\Pr(\eta;\theta)$;
      Create new sample: $\widetilde{\eta}_n^{(j)} = \begin{cases}\widetilde{\eta}'_n & \text{with probability } \alpha_n;\\ \widetilde{\eta}_n^{(j-1)} & \text{with probability } 1-\alpha_n.\end{cases}$
    Choose K nearly independent samples from $\{\widetilde{\eta}_n^{(j)}\}_{j=1}^J$ to form $\widetilde{\mathrm{H}} = \{\widetilde{\eta}_{kn}\}_{k=1}^K$;
    Prepare next chain: $\{\widetilde{\eta}_n^{(0)}\}_{n=1}^N = \{\widetilde{\eta}_n^{(J)}\}_{n=1}^N$
  Robbins-Monro Updating:
  Set gain value: $\gamma = \frac{s}{t}$ (or any $\gamma \equiv \gamma_t$ satisfying $\gamma_t \in (0, 1]$, $\sum_{t=1}^{\infty}\gamma_t = \infty$, and $\sum_{t=1}^{\infty}\gamma_t^2 < \infty$);
  Calculate gradient: $\frac{\partial}{\partial\theta}\ell(\widehat{\theta}; V, \widetilde{\mathrm{H}}) = \frac{\partial}{\partial\theta}\frac{1}{N}\sum_{n=1}^{N}\left[\frac{1}{K}\sum_{k=1}^{K}\ell(\widehat{\theta}; v_n, \widetilde{\eta}_{kn})\right]$;
  Update estimate: $\widehat{\theta} \leftarrow \widehat{\theta} + \gamma \times \frac{\partial}{\partial\theta}\ell(\widehat{\theta}; Y, \widetilde{\mathrm{H}})$;
  Update iteration count: $t \leftarrow t + 1$.

Algorithm 1 Metropolis–Hastings Robbins–Monro (MHRM) algorithm.

Vectorization on GPU and `xifa`

GPU and vectorized MHRM

GPU was originally designed for graphical processing. Today, GPU also serves as a tool for general purpose scientific computing. While CPU runs still much faster than a single GPU thread, the advantage of GPU computing lies in its high capacity of parallelization. In fact, most modern deep learning implementations highly rely on GPU (see CH12 in Goodfellow, Bengio, & Courville, 2016). According to *the GPU computing era*, "Today's GPUs use hundreds of parallel processor cores executing tens of thousands of parallel threads to rapidly solve large problems having substantial inherent parallelism." (Nickolls & Dally, 2010, p. 59). For example, NVIDIA GeForce RTX 2080 Ti, the local GPU used in our study, possesses 68 streaming multiprocessors (SMs), each SM containing 64 CUDA cores. Hence, a total of 4352 cores can be used for floating point operations.

Based on the powerfulness of GPU, we propose a vectorized version of the former MHRM algorithm for IFA. We call it vectorized MHRM (VMHRM) and its details can be found in Appendix A. The principle here is to stack data into higher order arrays and then use vectorized operations during computation. Besides the VMHRM itself, several practical considerations are also presented in Appendix B. These considerations include handling missing data, dealing with different numbers of categories, imposing simple parameter constraints, tuning jumping standard deviation, avoiding non-positive definite correlation matrices, and evaluating log-marginal likelihood .

`Python` package `xifa`

`xifa` is a `Python` package for accelerated item factor analysis. It is established on `Jax` (Bradbury et al., 2018), which uses the `XLA` (accelerated linear algebra) compiler for efficient array computation on GPUs. In addition, `Jax` provides a complete treatment for automatic differentiation allowing us to easily modularize our code for different IFA models. `xifa` supports IFA by GRM (Samejima, 1969) and GPCM (Muraki, 1992). The analysis can be either exploratory or confirmatory. Moreover, `xifa` is able to handle the presence of missing responses and unequal category items.

The design of `xifa` is highly motivated by `scikit-learn` (Pedregosa et al., 2011), a famous machine learning library in `Python`. An example `xifa` syntax for conducting GRM is

```
>>> from xifa import GRM                # import GRM from xifa
>>> grm=GRM(data=data, n_factors=5)     # create a GRM object
>>> grm.fit()                           # fit model to data
>>> grm.params["loading"]               # extract loading estimates
```

Here, `data` and `n_factors` are used to specify the data for analysis and the number of factors for exploratory IFA. Note that in this example the data set, also called `data`, is already prepared.

A complete tutorial with details for analyzing big-five personality data is available on GitHub.[1] The online material is a Jupyter notebook that can be interactively run on Colab. Thus, here we only mention several key points regarding using `xifa`. First, the data set must be an $N \times I$ `NumPy` array with integers coded from 0 to $C-1$, where C is the maximal number of ordered categories. Missing values must be represented by `nan` provided by `NumPy`. Second, `xifa` allows users to flexibly change hyperparameters for VMHRM. However, our simulations showed that the default setting generally performed well (see next section). Hence, in most cases users could simply use this default setting. In the fitting process, `xifa` prints the acceptance rate of MH samples and the value of minus complete data likelihood for each step. This information would be useful to monitor the convergence of VMHRM. If the algorithm unfortunately doesn't converge, one might (1) try more steps for warm up and RM updating; (2) change hyperparameters for MH sampling (e.g., jumping variance, or number of chains). Third, when N, I, and M are large (e.g., $N \geq 50000$, $I \geq 200$, $M \geq 20$), we may encounter GPU out-of-memory errors. An effective way to handle this error is to use the mini-batch approach described in Appendix B. To determine a "good" batch size, just try `batch_size` $=$ `round`(N/L) with $L = 2, 3, 4, \ldots$ and use the smallest L such that no GPU out-of-memory error arises.

[1] https://github.com/psyphh/xifa/blob/master/examples/ipip50.ipynb.

Numerical experiments

Overview of experiments

In this section, two numerical experiments are presented. Experiment A was designed to compare the performance of VMHRM implemented by `xifa` with *standard of care* methods under small number of factors ($M = 1, 3, 5$). In particular, we chose the BAEM and MHRM

performed by `mirt` (Chalmers, 2012), a popular `R` package for multidimensional IRT. Experiment B was designed to compare VMHRM with an IFA procedure for high-dimensional settings ($M = 10, 15, 20$). We chose IWAE because: (1) it performed well in the work (see Urban & Bauer, 2021); (2) its code is available in `PyTorch`, which can be run on GPU.

In both experiments, we varied the number of factors (M), the number of items (I), and the case size (N). These factors were mainly used to manipulate the degree of computational complexity. The number of items was set to $I = 5 \times M$ and $I = 10 \times M$. The case size was set to $N = 500, 1000, 2000, 4000$, and 8000. The levels of our manipulation partly covered the simulation design used in Chen et al. (2019) and Urban and Bauer (2021). As a result, there were $60 = 3 \times 2 \times 5$ settings considered in each experiment. For each setting, the number of replications was 100. Both experiments were conducted on a HP Z4 workstation with Intel Xeon W-2123 CPU (3.60 GHz), 32 GB RAM, and NVIDIA GeForce RTX 2080 Ti GPU.

For each replication, a data set was generated by GRM. Let 1_M denote an M-dimensional vector with all elements being one. An individual latent factor was first sampled through $\eta \sim \text{Normal}(0, \Phi)$, where $\Phi = 0.3 \times 1_M 1_M^T + 0.7 \times I_M$. The corresponding linear predictor was an $I \times (C+1)$ matrix $\tau = \mathrm{N} + \Lambda\eta$ with

$$\begin{aligned} \mathrm{N} &= 1_I \otimes (-\infty, -1, -.4, .4, 1, \infty)^T, \\ \Lambda &= I_M \otimes [1_K \otimes (2, 1.5, 1, 1.5, 2)], \end{aligned} \tag{7}$$

where $K = 5$, 10, and $\otimes$ denotes the Kronecker product. The ordinal response v was obtained by Eq. 2. These parameter values represent an ideal setting which assumes symmetric thresholds and large communalities ranged from .5 to .8. We believe that this ideal setting could reduce the occurrence of non-converged solution (e.g., Li, 2016). To make the comparison fair, the different algorithm implementations used the same data set for fitting.

The performance evaluation was based on three metrics: mean square error (MSE), average computational time (speed), and average number of iterations (efficacy). MSE evaluated the overall estimation quality through:

$$\widehat{MSE} = \frac{1}{100} \sum_{r=1}^{100} \frac{1}{P} \sum_{p=1}^{P} \left(\widehat{\vartheta}_p^{(r)} - \vartheta_p^* \right)^2, \tag{8}$$

where ϑ_p^* denotes the true value of the p^{th} model parameter, and $\widehat{\vartheta}_p$ denotes the corresponding estimate at the r^{th} replication.

Experiment A: Low-dimensional settings

Experiment A compared four implementations under $M = 1$, 3, and 5, including `xifa-VMHRM1`, `xifa-VMHRM5`, `mirt-MHRM`, and `mirt-BAEM`. The number after VMHRM indicates how many MH draws were used to approximate complete data likelihood. As mentioned before, VMHRM generated K samples by constructing K parallel chains, different from the Markov chain Monte Carlo (MCMC) practice. In theory, larger K results in better approximation for the complete data likelihood. For `mirt-MHRM`, only $K = 1$ was used, so we omitted the indicator after the name. Only non-zero loadings, the finite intercepts, and the factor correlations were estimated here. In other words, confirmatory IFA was considered.

Both VMHRM and MHRM were conducted through three stages: (1) 150 warmup steps for MCMC; (2) 200 StEM iterations to obtain a starting value for RM update; (3) at most 500 MHRM iterations for computing an MML estimate. In each StEM and MHRM iteration, there were four warmup steps for MH sampling. The gain sequence $\gamma_t = \frac{1}{t}$ was used for RM updating. BAEM used the number of quadrature points per dimension as 61, 15, and 7 for 1, 3, and 5 factor conditions, respectively. The maximal number of BAEM iterations was set as 700. For all implementations, the tolerance for declaring convergence was set as $\epsilon = 10^{-4}$. As VMHRM and MHRM are stochastic algorithms, they were considered to converge when three successive differences between the estimates were below the tolerance.

Figure 1 displays the evaluation metrics of Experiment A. The MSE of `xifa-VMHRM1`, `xifa-VMHRM5`, and `mirt-MHRM` were almost the same, suggesting equal quality estimates. However, the `mirt-BAEM` estimates only coincided with those of other methods under $M = 1$ and $M = 3$. For $M = 5$, `mirt-BAEM` resulted in higher MSE because of too few quadrature points per dimension. Note that using more quadrature points here is not a remedy either due to its extra computational time.

The average computational time indicates that `xifa-VMHRM1` and `xifa-VMHRM5` were reasonably fast, ranging from 3.45 s to 9.43 s across all conditions. They were the fastest under $M = 3$ and $M = 5$. Here, `xifa-VMHRM1` was slightly faster than `xifa-VMHRM5`. On the other hand, the average running time of `mirt-MHRM` and `mirt-BAEM` varied largely across conditions. The average running time of `mirt-MHRM` ranged from 4.69 s to 472.61 s, and for `mirt-BAEM`, it ranged from 0.11 s to 499.56 s. The `mirt-BAEM` was the fastest implementation for $M = 1$ (below 1 s), but the slowest for $M = 5$ (499.56 s).

The average number of iterations shows that `mirt-BAEM` was the most effective, followed by `xifa-VMHRM5`, `xifa-VMHRM1`, and `mirt-MHRM`. The lower number of iterations of `mirt-BAEM` is likely related to its deterministic nature. The efficacy of `xifa-VMHRM5` over `xifa-VMHRM1` is due to the higher number of MH draws for approximation. The inefficacy of `mirt-MHRM`

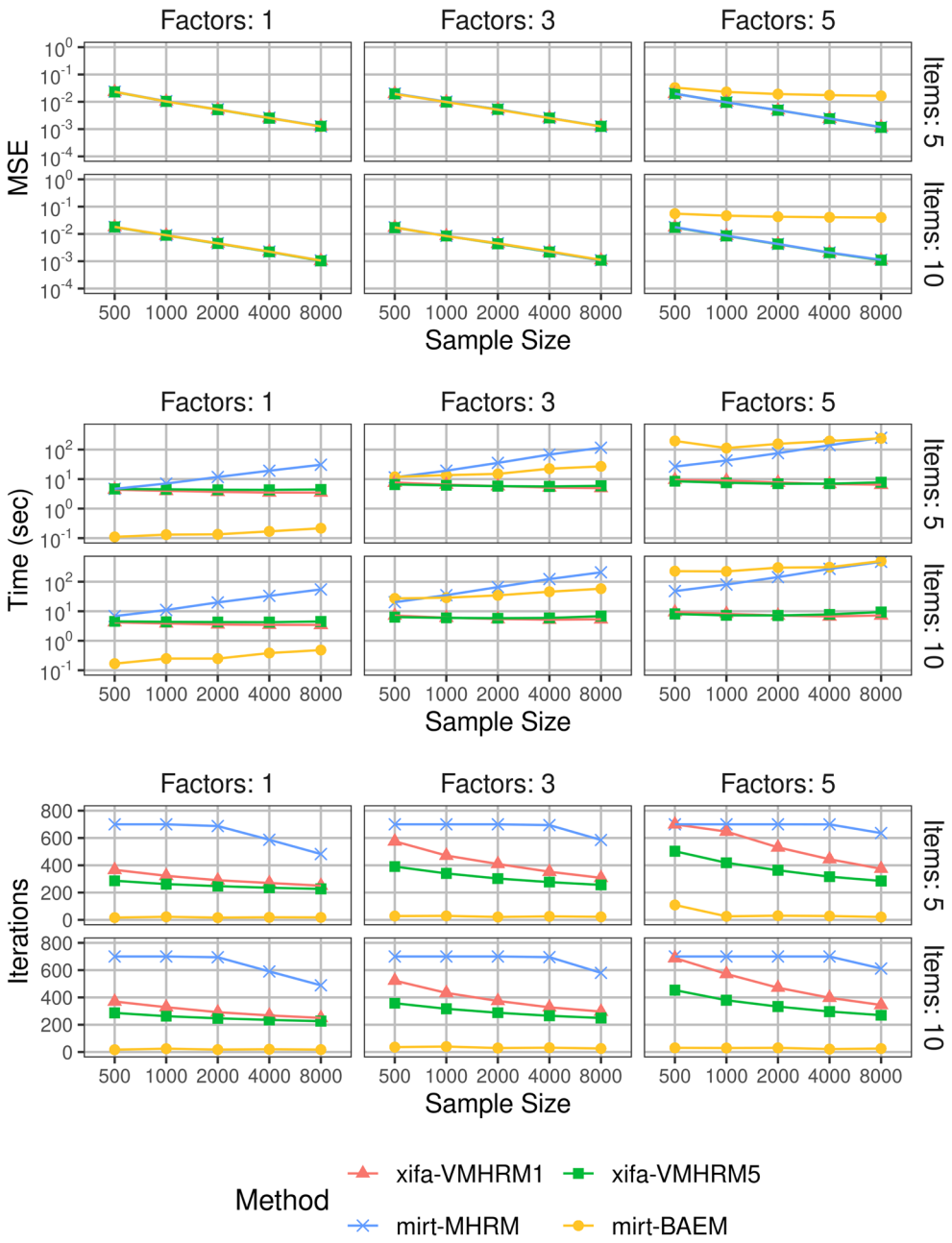

Fig. 1 Mean square error, average computational time, and average number of iterations for Experiment A

is unexpected. For most conditions, the average number of iterations for `mirt-MHRM` was 700, indicating the lack of convergence.[2] In spite of the lack of convergence, `mirt-MHRM` still yielded estimates with comparable MSE.

To better understand the acceleration due to GPU, we conducted two supplemental analyses. (1) For each condition, we calculated the ratio of average computational times per iteration between `mirt-MHRM` and `xifa-VMHRM1`. This metrics ignored that `mirt-MHRM` made more iterations. The five-number summary (i.e., minimum, first quartile, median, third quartile, and maximum) of these ratios across conditions was 0.57, 2.20, 5.53, 10.53, and 37.08, respectively. In general, GPU had more effect under larger conditions (with respect to M, I, and N). We only found `xifa-VMHRM1` slower than `mirt-MHRM` per iteration when $M = 1$ and $N \leq 1000$. (2) We ran `xifa-VMHRM1` on CPU with only ten replications, and calculated the ratio of average computational times between

[2]By default, the `mirt` sets the tolerance to 10^{-3} for MHRM. This looser criterion could yield better convergence results. However, it resulted in estimates with unacceptably high MSE. Hence, we decided to use $\epsilon = 10^{-4}$ for `mirt-MHRM` in the final experiment versions.

CPU and GPU implementations. The resulting five-number summary was 1.74, 5.13, 9.54, 15.99, and 28.72. The acceleration increased with either of M, I, and N. The GPU-implemented VMHRM was at least 7 times faster than its CPU version MHRM when $M \geq 3$ and $N \geq 2000$. Note that these computational times were determined from the same `Jax` code.

Experiment B: High-dimensional settings

Experiment B compared four implementations under $M = 10$, 15, and 20, including `xifa-VMHRM1`, `xifa-VMHRM5`, `torch-IWAE5`, and `torch-IWAE25`. The number after IWAE indicates how many IW samples were used. In theory, more IW samples result in more accurate approximation for marginal likelihood at the price of higher computational complexity. Because the available IWAE code was made for exploratory IFA, only factor loadings and intercepts were estimated in Experiment B. MSE was calculated from rotated results made by GEOMIN (Yates, 1987) via `R` package `GPArotation` (Bernaards & Jennrich, 2005). The maximal number of iterations for rotation was set to 5000. Note that the gradient projection algorithm (GPA; Jennrich, 2002) might not converge. Hence, for each rotation task, GPA was conducted repeatedly with different starting values up to 20 times. If no attempts converged, we chose the solution with the lowest GEOMIN function value.

The implementation of VMHRM was the same as in Experiment A. For IWAE, the implementation and the hyperparameter values were established mainly according to Urban and Bauer (2021). The maximal number of iterations was set to 700. Note that IWAE utilized the AMSGrad method (Reddi, Kale, & Kumar, 2018) that updated estimates with mini-batches of size 32. Hence, each iteration updated estimates `ceil`$(N/32)$ times.

Figure 2 displays the evaluation metrics of Experiment B. The four implementations resulted in quite different MSEs, except for two cases: (1) $M = 10$ and $I = 50$; (2) $M = 15$ and $I = 75$. Under other conditions, `torch-IWAE5` and `torch-IWAE25` resulted in much higher MSE than `xifa-VMHRM1` and `xifa-VMHRM5`. Unfortunately, IWAE seemed to yield inconsistent estimators when both M and I were large. For example, the MSE did not change by sample size when $M = 20$ and $I = 200$ were used. The inconsistency of IWAE might be fixed by tuning the model and optimization hyperparameters. However, it is a difficult and time consuming task.

The average computational time of `xifa-VMHRM1` and `xifa-VMHRM5` were the lowest among the four, ranging from 6.07 s to 27.73 s. The minimum corresponded to the smallest condition: $M = 10$, $I = 50$, $N = 500$. The maximum corresponded to the largest condition: $M = 20$, $I = 200$, $N = 8000$. Since `xifa-VMHRM1` sampled fewer MH draws than `xifa-VMHRM5`, it was slightly faster under most conditions. In contrast, `torch-IWAE5` and `torch-IWAE25` were much slower. Their average computational times ranged from 86.21 s to 206.46 s.

It is worth noting that `xifa-VMHRM5` was occasionally faster than `xifa-VMHRM1` even under a small sample size like $N = 500$. The reason is that `xifa-VMHRM5` used fewer iterations to finish the optimization task. This result demonstrates the usefulness of VMHRM on GPU even under small sample sizes. By sampling more MH draws, `xifa-VMHRM5` could yield a better approximation for complete data likelihood in each iteration. The average number of iterations were the highest among the four with `torch-IWAE25` and `torch-IWAE5` under $N = 500$. However, it decreased with N. Because the MSE of IWAE under large sample sizes became unacceptably high, it is meaningless to appreciate the smaller number of iterations of IWAE. Perhaps, the current convergence criterion of IWAE is too loose for large IFA models (e.g., $M \geq 15$ and $I \geq 150$). However, this speculation requires further experimentation to confirm.

To better understand the acceleration due to GPU, we also ran Experiment B on CPU with only ten replications, and calculated the ratio of average computational times between CPU and GPU implementations. For VMHRM, the five-number summary of the ratio was 6.42, 16.51, 25.84, 29.06, and 33.80, indicating large accelerations under most settings. In contrast, the summary for IWAE was only 0.28, 0.55, 0.70, 0.97, and 2.10. The GPU version was only faster than the CPU with larger M, I, and N. These results indicate that a GPU implementation alone is not sufficient for speed-up. The nature or the design of an algorithm is still important. By default, the IWAE code only processes a small batch of data for updating estimates. This mini-batch approach might reduce the capacity and effectiveness of parallelization. However, a more detailed analysis and experiment are required to understand this phenomenon.

A real data example

In this section, we demonstrate the powerfulness of VMHRM on GPU through a real data example. Two versions of IPIP-NEO (International Personality Item Pool NEO) data sets were used for demonstration (Johnson, 2015, 2018). The IPIP-NEO scale is composed of 300 items to measure 30 personality facets belonging to the Big Five traits — neuroticism, extroversion, openness, agreeableness, and conscientiousness. For each item, a personality description is presented (e.g.,

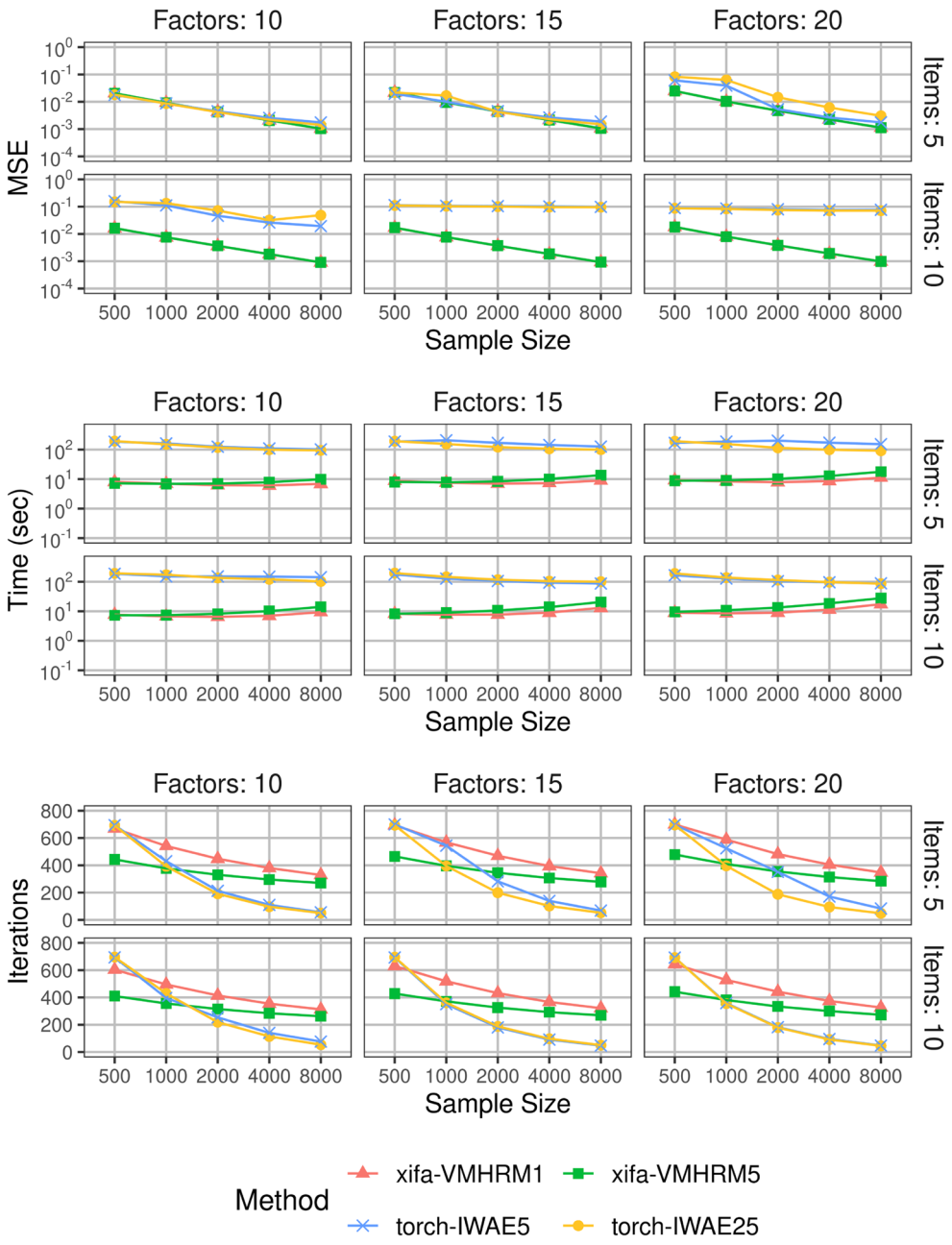

Fig. 2 Mean square error, average computational time, and average number of iterations for Experiment B

worries about things), and then the respondent is required to choose a degree of agreement from *very inaccurate*, *moderately inaccurate*, *neither accurate nor inaccurate*, *moderately accurate*, and *very accurate*. The first version, IPIPv1 included the responses of 20,993 subjects (Johnson, 2015). The second version, IPIPv2, included 307,313 subjects (Johnson, 2018). Zhang et al. (2020) used IPIPv1 to demonstrate their improved StEM.

We fit a 30-dimensional confirmatory GPCM model to both IPIPv1 and IPIPv2. Each item was assumed to be influenced by only one of the 30 personality facets. The correspondence between items and facets could be found in Johnson (2021). These personality facets were set to be correlated. To find MML estimates under IPIPv1 and IPIPv2, we used the same implementation described in our numerical experiments except that: (1) the factor correlation matrix was updated by an empirical method to avoid non-positive definiteness; (2) because of the large sample size of IPIPv2, an mini-batch approach was used to calculate the gradient with a batch size of 80000 (for details, see Appendix B). Code for the real data example is

Table 1 The marginal maximum likelihood estimate for factor loadings under IPIPv2

Subtraits	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
Neuroticism										
N1: Anxiety	1.32	1.07	1.21	2.01	1.36	1.00	1.04	0.80	0.83	0.83
N2: Anger	2.21	2.18	1.71	1.05	2.20	1.83	1.40	1.47	1.17	0.69
N3: Depression	1.70	2.31	2.19	2.10	0.90	1.35	0.77	0.96	1.92	1.72
N4: Self-consciousness	0.94	0.77	1.18	0.99	0.72	0.53	0.71	0.61	0.78	0.96
N5: Immoderation	0.57	0.56	0.63	0.73	0.41	0.95	0.88	0.98	0.40	0.51
N6: Vulnerability	1.53	1.26	1.37	0.72	1.27	1.39	0.74	1.28	0.82	1.34
Extroversion										
E1: Friendliness	1.36	1.02	2.09	1.76	0.95	0.74	1.05	1.46	0.57	1.08
E2: Gregariousness	1.30	1.10	1.07	0.76	0.68	1.12	1.10	1.41	1.65	1.05
E3: Assertiveness	1.68	1.38	0.68	0.56	1.50	1.20	1.06	0.74	0.63	0.67
E4: Activity Level	1.28	1.39	0.96	0.70	0.36	0.53	0.34	0.39	0.30	0.45
E5: Excitement-seeking	1.46	1.39	1.34	0.74	0.89	1.08	0.57	0.92	0.43	0.44
E6: Cheerfulness	1.26	1.77	0.56	0.80	1.37	1.32	0.82	0.76	0.60	0.45
Openness										
O1: Imagination	1.20	1.12	1.96	1.58	0.88	0.71	1.41	0.92	0.83	1.15
O2: Artistic Interests	1.96	0.54	0.98	0.49	1.02	2.58	0.28	1.39	0.24	0.43
O3: Emotionality	1.30	0.65	0.34	0.13	0.32	1.23	1.40	0.61	0.82	0.90
O4: Adventurousness	0.68	0.61	0.55	0.82	1.02	1.91	1.90	0.64	0.32	0.71
O5: Intellect	0.67	1.13	1.05	0.84	0.93	1.02	1.20	1.49	1.42	1.25
O6: Liberalism	0.76	0.37	0.79	0.33	0.86	0.49	0.75	0.89	1.18	0.38
Agreeableness										
A1: Trust	1.52	1.72	1.66	1.00	1.20	0.53	2.12	0.90	0.66	0.96
A2: Morality	0.40	0.58	0.42	1.06	0.63	1.30	0.61	0.66	1.70	0.95
A3: Altruism	1.07	0.88	1.38	1.49	0.67	0.77	1.05	0.68	1.00	1.17
A4: Cooperation	0.32	0.26	0.38	0.66	0.70	0.51	0.95	1.25	0.85	0.54
A5: Modesty	0.34	0.45	0.36	0.29	0.75	2.79	2.86	0.41	0.42	0.49
A6: Sympathy	0.97	1.24	0.00	0.62	0.82	0.84	0.34	0.70	0.59	0.49
Conscientiousness										
C1: Self-efficacy	1.17	1.32	1.46	0.90	1.25	1.62	0.79	0.82	0.98	0.61
C2: Orderliness	1.31	1.31	0.48	1.19	0.92	0.85	1.07	0.90	0.57	0.78
C3: Dutifulness	0.96	0.93	0.42	1.06	0.73	1.02	1.01	0.83	1.02	0.86
C4: Achievement-striving	0.78	1.59	0.97	0.93	1.27	0.67	0.40	0.63	1.13	1.16
C5: Self-discipline	0.96	0.88	1.51	1.20	1.26	1.61	1.22	1.32	1.77	0.74
C6: Cautiousness	0.46	0.50	0.34	1.85	1.42	0.86	1.65	0.75	2.22	0.61

Note that V1, V2,..., V10 are not the labels for the 300 IPIP items. They represent the ten prespecified indicators for the corresponding subtraits

available as supplemental material.[3] The analyses were run on Kaggle (https://www.kaggle.com), a platform for data science. Fantastically, Kaggle provided GPU with 16 GB memory, exceeding our local GPU machine.

With a GPU on Kaggle, the VMHRM used 30 s to finish the optimization tasks under IPIPv1. The same took 16 min under IPIPv2. As a comparison, the improved StEM used 32 min to find an estimate based on only the 7325 complete data cases in IPIPv1 (Zhang et al., 2020). The VMHRM did not drop any incomplete case due to the full information approach we introduced to handle missing values. Since the sample size of IPIPv2 is large, we only

[3] https://github.com/psyphh/xifa/blob/master/examples/ipip300v1.ipynb; https://github.com/psyphh/xifa/blob/master/examples/ipip300v2.ipynb.

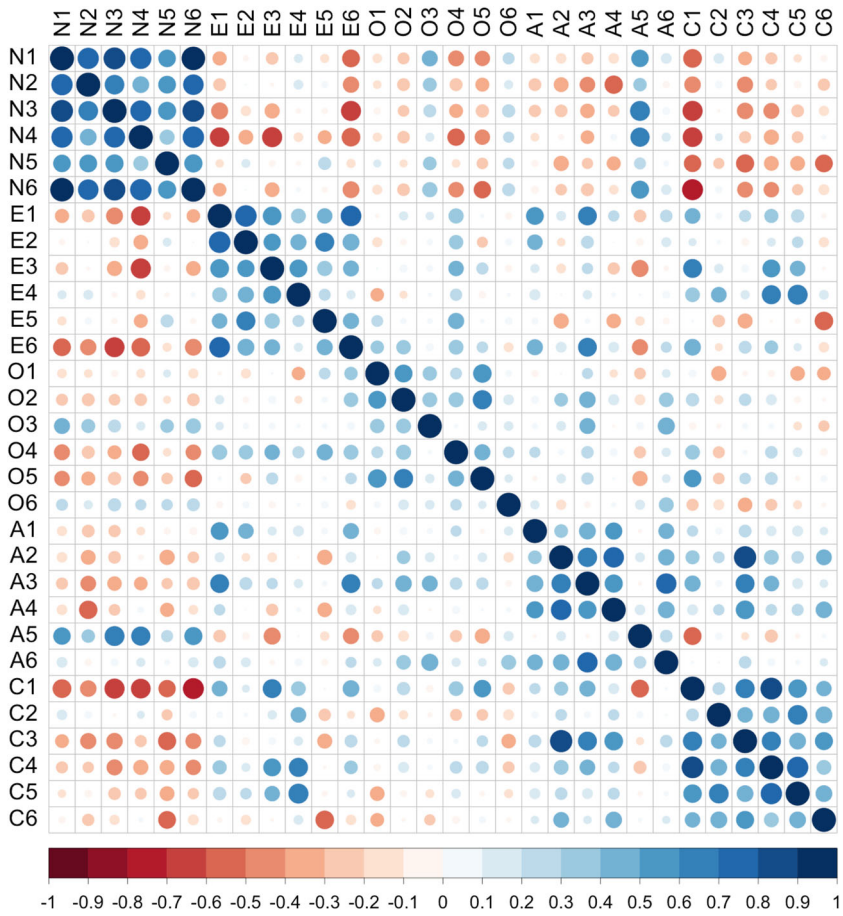

Fig. 3 Visualization of the estimated factor correlation matrix under IPIPv2

present its corresponding MML estimate. Table 1 shows the estimates for the non-zero loadings, and Fig. 3 visualizes the estimated factor correlation matrix. Their relative values are quite similar to those found in Zhang et al. (2020).

The log-marginal likelihood under the MML estimate was evaluated by Monte Carlo integration with 5000 quadrature points. To avoid out of memory problems, the batch size was set to 128. The evaluation took 8 s for IPIPv1 and 205 s for IPIPv2. The likelihood value was −388.535 for IPIPv1 and negative infinity for IPIPv2. We found that the 53, 557^{th} and 103, 686^{th} cases corresponded to negative infinity. It seems that the two cases correspond to "weird" response patterns. The log-marginal likelihood without these two cases was −391.606.

Discussion

MML estimation in IFA is a challenging task under high dimensionality. In this study, we proposed a VMHRM algorithm for both the GRM and the GPCM. The algorithm is implemented by `xifa`, a `Python` package. Our numerical experiments demonstrated that VMHRM on GPU may run 33 times faster than its CPU implementation. The medians of acceleration were nine times for dimensions $\leq$ 5, and 25 times for dimensions $\geq$ 10. The degree of acceleration increases with the number of factors, the sample size, and the number of items. Therefore, the GPU-implemented VMHRM is most appropriate for high-dimensional IFA with large data sets. Even for small data sets, VMHRM is useful for handling more MH samples.

When the number of factors is at least five, the GPU-implemented VMHRM is much faster than existing IFA implementations such as the BAEM, MHRM implemented by `mirt`, and the IWAE. With 20 factors, 200 items, and 8000 cases, VMHRM used about 28 s to finish the optimization task. This progress in computation time is a breakthrough for high-dimensional IFA. With the help of GPU, it is possible to use MML for many large-scale psychological and educational applications.

The fact that GPU acceleration exists does not imply that every code could be accelerated on GPU without

modification. For example, the IWAE code written by Urban and Bauer (2021) is surprisingly slower on GPU under most conditions. The degree of GPU acceleration depends on whether related operations can be parallelized. We have seen that GPU takes advantage of larger input arrays. Otherwise, a CPU-based serial computation remains faster.

VMHRM is not limited to rely on GRM or GPCM. Other types of IFA models (e.g., item response tree, Bockenholt, 2012) could be used after replacing Eqs. 11 and 12 by category response functions. The deep learning libraries would automatically calculate the gradients via automatic differentiation. As a result, VMHRM would still compute the corresponding MML estimates. Our proposed algorithm could even be modified by other sampler and updating methods. It is possible to design new algorithms using the No-U-Turn Sampler (NUTS; Hoffman & Gelman, 2014), Polyak-Ruppert averaging (Polyak & Juditsky, 1992; Ruppert, 1988), and stochastic proximal methods (Zhang & Chen, 2021). It would be interesting to compare the empirical performance of VMHRM with these new algorithms.

The major limitation of the present work is that VMHRM was only evaluated under very restricted simulation settings. Our simulation uses a GRM model with "ideal" parameter values to generate data. In addition, the simulation only considers simple stochastic gradient descent with $\gamma_t = \frac{1}{t}$ after a fixed number of StEM steps. It would be interesting to see the behavior of VMHRM under $\gamma_t = \frac{1}{t^\alpha}$ for $\alpha \in (0.5, 1)$ after an adaptive number of StEM steps (e.g., Zhang et al., 2020). With the acceleration made by GPU, it is possible to evaluate MML algorithms for IFA in a more comprehensive manner.

We believe that many psychometric modeling methods would also benefit from GPU provided that their relevant operations are appropriately vectorized. For example, Bayesian psychometric models (e.g., Edwards, 2010; Muthén & Asparouhov, 2012) could be speeded up by constructing parallel Markov chains as we have done it with our VMHRM. We expect that GPU computing will play a central role in large-scale psychometric modeling in the near future.

Open practices statements

The source code of the software package and real data example are available at https://github.com/psyphh/xifa. Data derived from public domain resources. None of the experiments were preregistered.

Appendix A: The VMHRM algorithm

Before introducing the VMHRM, we first go through the notations. Recall that $v_n = (v_{n1}, v_{n2}, ..., v_{nI})$ denotes the response vector of n^{th} subject on I polytomous items. The event $v_{ni} = c$ can be represented by a C-dimensional one-hot encoding vector y_{ni}, that is,

$$v_{ni} = c \iff y_{ni} = (\underbrace{0, ..., 0}_{c\ 0\text{'s}}, 1, \overbrace{\underbrace{0, ..., 0}_{C-c-1\ 0\text{'s}}}^{C\ \text{elements}}). \tag{9}$$

In other words, the $(c+1)^{th}$ element of y_{ni} is used to indicate whether the event $v_{ni} = c$ occurs. Hence, the random sample V can be equivalently represented by Y, an $N \times I \times C$ array with $y_n = (y_{n1}, y_{n2}, ..., y_{nI})$ being its n^{th} element. In addition, the log-probability of v_{ni} given η_n can be expressed in inner product form:

$$\begin{aligned} \log \Pr(y_{ni}|\eta_n) &= \sum_{c=0}^{C-1} y_{nic} \log \pi_{ic}(\eta_n) \\ &= y_{ni}^T \log \pi_i(\eta_n). \end{aligned} \tag{10}$$

In each MH step, we sample K valid draws for each observation. Hence, the MH samples can be represented by a $K \times N \times M$ array $\widetilde{\mathrm{H}}$ with $\widetilde{\eta}_{knm}$ being its (k, n, m) element. The intercepts ν_{ic} are packed into N, which is an $I \times C^*$ matrix, where $C^* = C + 1$ for GRM and $C^* = C$ for GPCM. Note that N is the uppercase form of ν so that it is different from N, the sample size. The loadings λ_{im} are placed into Λ, which is an $I \times M$ matrix. The correlation of η is already a $M \times M$ matrix, denoted by Φ. The three model parameter matrices can be joined into $\theta = \{\mathrm{N}, \Lambda, \Phi\}$. Here, θ is not a vector anymore. θ can be understood as a container in programming languages, such as `list` and `dict` in `Python`. Note that most psychometric works use θ to pack the freely estimated elements of N, Λ, and Φ. That approach would result in unnecessary data manipulation for us regarding reformatting parameters and their gradients.

We separate the computations related to VMHRM algorithm into four main parts: (1) computing category response probabilities, (2) evaluating complete data log-likelihood, (3) sampling MH draws, and (4) updating parameter estimates by RM. Because standard linear algebra only handles 1D and 2D array computations, we will use functions in `NumPy` (Harris et al., 2020) and the rule of broadcasting (The NumPy Community, 2021) to express higher order array operations in the following discussion. It should be noted that the rule of broadcasting plays a central role in vectorization with deep learning libraries.

1. To calculate category response probabilities, we introduce the linear predictor $\tau_{knic} = \nu_{ic} \pm \lambda_i^T \widetilde{\eta}_{kn}$ for each combination of k, n, i, c. For GRM, these quantities can be computed by the following vectorized operations:

$$\underbrace{\mathrm{T}}_{K\times N\times I\times(C+1)} = -\underbrace{\mathrm{N}}_{I\times(C+1)} - \underbrace{\mathtt{expand\_dims}_4\left(\overbrace{\left(\underbrace{\widetilde{\mathrm{H}}}_{K\times N\times M}\ \underbrace{\Lambda^T}_{M\times I}\right)}^{K\times N\times I}\right)}_{K\times N\times I\times 1}, \tag{11}$$

where $\mathtt{expand\_dims}_4(\cdot)$ adds a new dimension (or axis) to the fourth axis of $\widetilde{\mathrm{H}}\Lambda^T$. Note that $\mathtt{expand\_dims}_4(\cdot)$ doesn't change the actual data in $\widetilde{\mathrm{H}}\Lambda^T$. Let Π denote a $K\times N\times I\times C$ array with $\pi_{ic}(\widetilde{\eta}_{kn})$ being its (k, n, i, c) element. The GRM-implied Π can be obtained from T by:

$$\underbrace{\Pi}_{K\times N\times I\times C} = \mathtt{diff}_4\left(\mathtt{expit}\left(\underbrace{\mathrm{T}}_{K\times N\times I\times(C+1)}\right)\right). \tag{12}$$

Here, $\mathtt{expit}(\cdot)$ is an element-wise operator that transforms τ_{knic} to $\frac{1}{1+\exp(-\tau_{knic})}$, while $\mathtt{diff}_4(\cdot)$ is the lagged difference operator along the fourth axis of the input array, i.e., $\mathtt{diff}_4(\cdot)$ calculates $\frac{1}{1+\exp(-\tau_{knic})} - \frac{1}{1+\exp(-\tau_{kni(c+1)})}$. For GPCM, we can obtain the cumulative sum of linear predictors via the expression:

$$\underbrace{\mathrm{T}^*}_{K\times N\times I\times C} = \overbrace{\mathtt{cumsum}_2\left(\underbrace{\mathrm{N}}_{I\times C}\right)}^{I\times C} + \underbrace{\overbrace{\mathtt{expand\_dims}_4\left(\widetilde{\mathrm{H}}\Lambda^T\right)}^{K\times N\times I\times C}\times\underbrace{d}_{C}}_{K\times N\times I\times 1}, \tag{13}$$

where $d = (0, 1, ..., C-1)$ is a scoring vector, and $\mathtt{cumsum}_2(\cdot)$ calculates the cumulative sum along the second axis. The GPCM-implied response probabilities equal:

$$\underbrace{\Pi}_{K\times N\times I\times C} = \underbrace{\exp(\mathrm{T}^*)}_{K\times N\times I\times C} / \overbrace{\mathtt{expand\_dims}_4\left(\underbrace{\mathtt{sum}_4\left(\exp(\mathrm{T}^*)\right)}_{K\times N\times I}\right)}^{K\times N\times I\times 1}. \tag{14}$$

2. To evaluate complete data log-likelihoods, we first target individual likelihood values, that is, for each pair of k and n, we compute:

$$\ell(\theta; y_n, \widetilde{\eta}_{kn}) = \sum_{i=1}^{I} y_{ni}^T \log \pi_i(\widetilde{\eta}_{kn}) - \tfrac{1}{2}\left[M\log(2\pi) - \log|\Phi| - \widetilde{\eta}_{kn}^T\Phi^{-1}\widetilde{\eta}_{kn}\right]. \tag{15}$$

These individual values are necessary for (1) obtaining the value of overall complete data log-likelihood; and (2) going through the acceptance rule in MH sampling. Let $\ell_{\mathrm{ind}}(\theta; Y, \widetilde{\mathrm{H}})$ denote the $K\times N$ matrix of individual complete data log-likelihoods. It can be expressed through the following expression:

$$\begin{aligned}\underbrace{\ell_{\mathrm{ind}}(\theta; Y, \widetilde{\mathrm{H}})}_{K\times N} = {} & \underbrace{\mathtt{sum}_{3,4}\left(\overbrace{\left(\underbrace{Y}_{N\times I\times C}\times\underbrace{\log(\Pi)}_{K\times N\times I\times C}\right)}^{K\times N\times I\times C}\right)}_{K\times N} \\ & - \underbrace{\tfrac{M}{2}\log 2\pi}_{\text{scalar}} - \underbrace{\tfrac{1}{2}\log|\Phi|}_{\text{scalar}} \\ & - \underbrace{\tfrac{1}{2}\mathtt{sum}_3\left(\mathtt{square}\left(\overbrace{\left(\underbrace{\widetilde{\mathrm{H}}}_{K\times N\times M}\ \underbrace{L^{-1}}_{M\times M}\right)}^{K\times N\times M}\right)\right)}_{K\times N},\end{aligned} \tag{16}$$

where L^{-1} is the Cholesky factor of Φ^{-1}, $\mathtt{square}(\cdot)$ is an element-wise operator for square and $\mathtt{sum}_{\mathtt{i,j}}(\cdot)$ calculates sums over i^{th} and j^{th} axes. The first term in Eq. 16 is a vectorzied expression for $\Pr(y_n|\eta_{km})$. The last term calculates the Mahalanobis distances of η_{kn}'s from the mean, the central part to calculate multivariate normal density. The overall likelihood can be obtained by averaging $\ell_{\mathrm{ind}}(\theta; Y, \widetilde{\mathrm{H}})$ over the first and second dimensions:

$$\underbrace{\ell(\theta; Y, \widetilde{\mathrm{H}})}_{\text{scalar}} = \mathtt{mean}\left(\underbrace{\ell_{\mathrm{ind}}(\theta; Y, \widetilde{\mathrm{H}})}_{K\times N}\right). \tag{17}$$

3. In order to get K many draws during parallel MH sampling, we can establish K independent Markov chains. This approach is more efficient when fewer per-iteration warmup steps are required. Let $\widetilde{\mathrm{Z}}$ denote a $K\times N\times M$ array with its elements $\widetilde{\zeta}_{knm}$ being independent samples from Normal$(0, \sigma^2_{\mathrm{jump}})$:

$$\underbrace{\widetilde{\mathrm{Z}}}_{K\times N\times M} \sim \mathtt{normal}_{\mathtt{K,N,M}}\left(0, \sigma^2_{\mathrm{jump}}\right). \tag{18}$$

Let $\widetilde{\mathrm{H}}' = \widetilde{\mathrm{H}} + \widetilde{\mathrm{Z}}$. Then, the acceptance probability for each element of $\widetilde{\mathrm{H}}'$ can be expressed as:

$$\underbrace{\mathrm{A}}_{K\times N} = \mathtt{minimum}\left(\underbrace{\ell_{\mathrm{ind}}(\theta; Y, \widetilde{\mathrm{H}}')/\ell_{\mathrm{ind}}(\theta; Y, \widetilde{\mathrm{H}})}_{K\times N}, \underbrace{1}_{\text{scaler}}\right), \tag{19}$$

and the updated MH sample for the next step can be expressed as:

$$\underbrace{\widetilde{\mathrm{H}}}_{K\times N\times M} = \texttt{where}\left(\underbrace{\mathrm{B}, \widetilde{\mathrm{H}}', \widetilde{\mathrm{H}}}_{K\times N\times M}\right), \tag{20}$$

where $\texttt{minimum}(\cdot,\cdot)$ returns the element-by-element minimum of two arrays, B is a $K \times N \times M$ array with 0-1 elements β_{knm} indicating whether η'_{knm} should be accepted, and $\texttt{where}(\cdot,\cdot,\cdot)$ is an element-by-element if-else statement. The condition array B can be constructed by sampling a Bernoulli distribution with success probability A. Note that B must be constructed in such a way that the elements in η'_{kn} and η_{kn} be simultaneously selected.

4. The final step is to update the parameter estimate. Let $\frac{\partial}{\partial\theta}\widehat{\ell} \equiv \frac{\partial}{\partial\theta}\ell(\widehat{\theta}; Y, \widetilde{\mathrm{H}})$ denote a container with elements as the gradients for parameter matrices, i.e., $\frac{\partial}{\partial\theta}\widehat{\ell} = \left\{\frac{\partial}{\partial \mathrm{N}}\widehat{\ell}, \frac{\partial}{\partial\Lambda}\widehat{\ell}, \frac{\partial}{\partial\Phi}\widehat{\ell}\right\}$. The elements of this gradient container can be calculated via automatic differentiation in deep learning libraries. The formula for model parameter update is the following:

$$\widehat{\theta} \leftarrow \widehat{\theta} + \gamma \times \frac{\partial}{\partial\theta}\widehat{\ell}. \tag{21}$$

Here, $+$ and $\times$ are interpreted in element-by-element manner. Note that current libraries for GPU computing do not have built-in vectorized operations between standard containers. Hence, $\widehat{\mathrm{N}}$, $\widehat{\Lambda}$, and $\widehat{\Phi}$ are updated sequentially, while the elements of each parameter matrix are updated in parallel. In addition, AD does not take into account the symmetry of Φ. Hence, before updating Φ, we need to adjust $\frac{\partial}{\partial\Phi}\widehat{\ell}$ by

$$\frac{\partial}{\partial\Phi}\widehat{\ell} \leftarrow \frac{\partial}{\partial\Phi}\widehat{\ell} + \frac{\partial}{\partial\Phi^T}\widehat{\ell} - \frac{\partial}{\partial\Phi}\widehat{\ell} \times I_M, \tag{22}$$

where I_M denotes the $M \times M$ identity matrix (see Corollary 1 in McCulloch, 1982).

Appendix B: Practical considerations for VMHRM

In this appendix, we present several practical considerations to optimize the performance and applicability of the VMHRM algorithm.

Handling missing data When conducting IFA, missing data may be encountered by either chance (e.g., careless response) or design (e.g., adaptive testing). In `Python`, missing response is represented by a special data type `nan` defined in `NumPy`. To vectorize MHRM in the presence of missing data, we simply fill the corresponding one-hot encoding vector with C many zero components:

$$v_{ni} = \texttt{nan} \iff y_{ni} = \overbrace{(\underbrace{0, 0, ..., 0}_{C\ 0\text{'s}})}^{C\ \text{elements}}.$$

As a result, the log-probability for $v_{ni} = \texttt{nan}$ given any level of η_n will be zero by virtue of $\log \Pr(y_{ni}|\eta_n) = y_{ni}^T \log \pi_i(\eta_n)$. The vectorized computation for log-likelihood can be used as described in Eq. 16. If the missing mechanism satisfies the so-called missing at random condition, the full information approach here could still yield a consistent estimate (Rubin, 1976).

Dealing with different numbers of categories Sometimes, we encounter the situation that the numbers of categories for the test items are different, that is, $C_i \neq C$ for some i. In this situation, we could still represent the whole responses (v_i) and intercepts (v_{ic}) by an $I \times C$ one-hot encoding matrix y and an $I \times C^*$ intercept matrix N, respectively, where $C^* = C + 1$ for GRM and $C^* = C$ for GPCM with $C = \max\{C_i\}_{i=1}^I$. However, for any item i such that $C_i < C$, the i^{th} row of y and N should be respectively padded by (1) $y_{ic} = 0$ if $c \geq C_i$; and by (2a) $v_{ic} = -\infty$ if $c \geq C_i$ for GRM; or (2b) $v_{ic} = 0$ if $c \geq C_i$ for GPCM. By this construction, the vectorized computation for log-likelihood as described in Eq. 16 remains valid. This construction implies that $\pi_{ic}(\eta) = 0$ and $\frac{\partial}{\partial v_{ic}}\widehat{\ell} = 0$ for $c \geq C_i$, thus the estimate for a fixed threshold does not change when updating model parameters.

Imposing simple parameter constraints So far, all loadings and factor correlations were freely estimated. In other words, only the so-called exploratory IFA was considered. In practice, we may restrict some elements in N, Λ, and Φ to zero or other values, which leads to the so-called confirmatory IFA. Let m_Λ denote an $I \times M$ mask matrix with only 0 (false) and 1 (true) logical values such that the (i, j) element of m_Λ is an indicator whether the (i, j) element of Λ is freely estimated. Similarly, we can construct m_N and m_Φ for N and Φ, respectively, and then put all the mask matrices into a container $m_\theta = (m_\mathrm{N}, m_\Lambda, m_\Phi)$. To restrict the values of parameters, the updating part in Eq. 21 can be modified as:

$$\widehat{\theta} \leftarrow \widehat{\theta} + \gamma \times m_\theta \times \frac{\partial}{\partial\theta}\widehat{\ell}.$$

Hereby, the mask matrices are used to mask the corresponding gradients so that only freely estimated parameters be updated. As a result, the estimates will stay in the constrained parameter space once the starting parameters satisfy the imposed constraints.

Tuning jumping standard deviation In Metropolis–Hastings sampling, a value σ_{jump} for jumping standard deviation must be specified. The smaller σ_{jump}, the higher is the rate for accepting a new Metropolis–Hasting candidate. It was suggested that 0.23 is an optimal acceptance rate for ($M > 5$; Gelman et al., 2013). To control the acceptance rate around a target value, the jumping variance can be adjusted adaptively in each warm-up iteration by $\sigma_{\text{jump}} \leftarrow \sigma_{\text{jump}} \pm \delta$, where δ is a preset number. This simple adjustment can effectively lead towards the optimal acceptance rate within 100 iterations if σ_{jump} is initialized as $2.4/\sqrt{M}$ and δ is set as 0.01. Note that the value of σ_{jump} can only be adjusted during the warmup stage. Otherwise, the Markov chain might not converge to its target distribution (Vihola, 2012).

Avoiding non-positive definite correlation matrices Under higher dimensionality (e.g., $M = 20$) and highly correlated factors (e.g., $\rho_{mm'} = 0.8$ for many pairs m, m'), our experience shows that line search methods often result in non-positive definite (NPD) estimates $\widehat{\Phi}$ for the correlation matrix. This NPD problem can crush the training process. To avoid the NPD problem, we can estimate $\widehat{\Phi}$ via RM update and an empirical correlation matrix of MH samples. More specifically, $\widehat{\Phi}$ can be updated by:

$$\widehat{\Phi} \leftarrow (1-\gamma)\widehat{\Phi} + \gamma R,$$

where R is an empirical correlation matrix based on the current $\widetilde{\mathrm{H}}$. By the normality of the latent factors, R maximizes complete-data likelihood in each iteration under standardized $\widetilde{\mathrm{H}}$. The update ensures the positive definiteness of $\widehat{\Phi}$. Note that this empirical method is only valid when all correlations are freely estimated.

Analyzing large data sets It is possible to conduct IFA on large data sets with millions or even billions of observations. When implementing VMHRM, the most memory-consuming task is to calculate the gradients. In principle, the gradient container for $\frac{\partial}{\partial\theta}\widehat{\ell}$ stores $I \cdot M$ (loadings) + $I(C+1)$ (intercepts) + M^2 (factor correlations) floating-point numbers in memory. Note that the gradient container is the average of $N \cdot K$ individual gradients: $\frac{\partial}{\partial\theta}\widehat{\ell} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{K}\sum_{k=1}^{K}\frac{\partial}{\partial\theta}\ell(\widehat{\theta}; y_n, \widetilde{\eta}_{kn})$. Therefore, to calculate $\frac{\partial}{\partial\theta}\widehat{\ell}$ in parallel, $N \cdot K$ units of memory is required, which may cause out-of-memory problem when N is large. A simple way to deal with large data sets is to use a mini-batch approach. That is, we can split both Y and $\widetilde{\mathrm{H}}$ into B many batches, say $\{(Y_{(b)}\widetilde{\mathrm{H}}_{(b)})\}_{b=1}^{B}$ along their first and second axis, and then update the estimate sequentially in b for each iteration:

$$\widehat{\theta} \leftarrow \widehat{\theta} + \gamma \times \frac{\partial}{\partial\theta}\ell(\widehat{\theta}; Y_{(b)}, \widetilde{\mathrm{H}}_{(b)}).$$

To maximize the degree of parallelism, we suggest using a large batch size such that the GPU memory can afford to compute $\frac{\partial}{\partial\theta}\ell(\widehat{\theta}; Y_{(b)}, \widetilde{\mathrm{H}}_{(b)})$. Note that the sample size in each batch is possibly different.

Evaluating log-marginal likelihood In high-dimensional settings, the log-marginal likelihood is often evaluated by Monte Carlo integration methods under the MML estimate $\ell(\widehat{\theta}; Y)$. Let $\Xi = (\xi_q)_{q=1}^{Q}$ denote a $Q \times M$ matrix with ξ_q generated from Normal$(0, \widehat{\Phi})$ randomly. We can approximate the likelihood in parallel via:

$$\ell(\theta; Y) \approx \texttt{mean}\left(\texttt{log}\left(\texttt{mean}_2\left(\texttt{prod}_{3,4}\overbrace{\left(\texttt{power}\left(\Pi, \underbrace{\texttt{expand\_dims}_2(Y)}_{N\times 1\times I\times C}\right)\right)}^{N\times Q\times I\times C}\right)\right)\right),$$

where Π is a $Q \times I \times C$ array with $\pi_{ic}(\xi_q)$ being its (q, i, c) element. Because of numerical errors in floating point arithmetic and $0 < \pi_{ic}(\xi_q) < 1$, the double product may cause $\ell(\widehat{\theta}; Y)$ to be evaluated as negative infinity if the model fits some data point extremely poorly. Note that the formula for $\ell(\theta; Y)$ involves a matrix of size $N \times Q \times I \times C$. To precisely evaluate the likelihood, Q is often set to a large number such as $Q = 5000$ (e.g., González, Tuerlinckx, De Boeck, & Cools, 2006). Hence, when N is large, the mini-batch approach described in the previous paragraph is necessary to avoid out-of-memory problems.

Funding This research was supported under grant number MOST109-2410-H-003-136-MY3 by the Ministry of Science and Technology in Taiwan.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Retrieved from https://www.tensorflow.org/.

Bernaards, C. A., & Jennrich, R. I. (2005). Gradient projection algorithms and software for arbitrary rotation criteria in factor

analysis. *Educational and Psychological Measurement*, *65*, 676–696.
Bock, R. D., & Aitkin, M. (1981). Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm. *Psychometrika*, *46*(4), 443–459.
Bock, R. D., Gibbons, R., & Muraki, E. (1988). Full-information item factor analysis. *Applied Psychological Measurement*, *12*(3), 261–280.
Bock, R. D., & Lieberman, M. (1970). Fitting a response model for n dichotomously scored items. *Psychometrika*, *35*(2), 179–197.
Bockenholt, U. (2012). Modeling multiple response processes in judgment and choice. *Psychological Methods*, *17*(4), 665–678.
Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., & Maclaurin, D. (2018). JAX: Composable transformations of Python+NumPy programs.
Cai, L. (2010a). High-dimensional exploratory item factor analysis by a Metropolis–Hastings Robbins–Monro algorithm. *Psychometrika*, *75*, 33–57.
Cai, L. (2010b). Metropolis–Hastings Robbins–Monro algorithm for confirmatory item factor analysis. *Journal of Educational and Behavioral Statistics*, *35*(3), 307–335.
Cai, L. (2017). *flexMIRT®: Flexible multilevel multidimensional item analysis and test scoring*. Chapel Hill, NC: Vector Psychometric Group.
Cai, L., Du Toit, S., & Thissen, D. (2011). *IRTPRO: Flexible, multidimensional, multiple categorical IRT modeling*. Chicago: Scientific Software International.
Chalmers, R. P. (2012). mirt: A multidimensional item response theory package for the R environment. *Journal of Statistical Software*, *48*(6), 1–29.
Chen, Y., Li, X., Liu, J., & Ying, Z. (2021). *Item response theory – a statistical framework for educational and psychological measurement*.
Chen, Y., Li, X., & Zhang, S. (2019). Joint maximum likelihood estimation for high-dimensional exploratory item factor analysis. *Psychometrika*, *84*(1), 124–146.
Chen, Y., Li, X., & Zhang, S. (2020). Structured latent factor analysis for large-scale data: Identifiability, estimability, and their implications. *Journal of the American Statistical Association*, *115*(532), 1756–1770.
Cho, A. E., Wang, C., Zhang, X., & Xu, G. (2020). Gaussian variational estimation for multidimensional item response theory. *British Journal of Mathematical and Statistical Psychology*.
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 (Methodological)*, *39*(1), 1–38.
Edwards, M. C. (2010). A Markov chain Monte Carlo approach to confirmatory item factor analysis. *Psychometrika*, *75*(3), 474–497.
Fahrmeir, L., & Tutz, G. (1994). *Multivariate statistical modelling based on generalized linear models*. New York: Springer-Verlag.
Gelman, A., Carlin, J., Stern, H., Dunson, D., Vehtari, A., & Rubin, D. (2013). *Bayesian data analysis*, (3rd ed.). New York: Taylor & Francis.
Gibbons, R. D., & Hedeker, D. R. (1992). Full-information item bi-factor analysis. *Psychometrika*, *57*(3), 423–436.
González, J., Tuerlinckx, F., De Boeck, P., & Cools, R. (2006). Numerical integration in logistic-normal models. *Computational Statistics & Data Analysis*, *51*(3), 1535–1548.
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge: MIT Press. http://www.deeplearningbook.org.
Harris, C. R., Millman, K. J., Walt, S. J., van der Gommers, R., Virtanen, P., & Cournapeau, D. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.
Hoffman, M. D., & Gelman, A. (2014). The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *15*(1), 1593–1623.
Hui, F. K. C., Warton, D. I., Ormerod, J. T., Haapaniemi, V., & Taskinen, S. (2017). Variational approximations for generalized linear latent variable models. *Journal of Computational and Graphical Statistics*, *26*(1), 35–43.
Jackson, D. L., Gillaspy Jr, J. A., & Purc-Stephenson, R. (2009). Reporting practices in confirmatory factor analysis: an overview and some recommendations5. *Psychological Methods*, *14*(1), 6.
Jennrich, R. (2002). A simple general method for oblique rotation. *Psychometrika*, *67*(1), 7–19.
Johnson, J. A. (2015). Data from Johnson, J. A. (2005). Ascertaining the validity of web-based personality inventories. *Journal of Research in Personality*, *39*, 103–129. OSF. Retrieved from osf.io/sxeq5.
Johnson, J. A. (2018). Data from Johnson, J. A. (2014). Measuring thirty facets of the five factor model with a 120-item public domain inventory: Development of the IPIP-NEO-120. *Journal of Research in Personality*, *51*, 78–89. OSF. osf.io/wxvth.
Johnson, J. A. (2021). Scoring key for the ipip-neo-300 and ipip-neo-120. OSF. osf.io/ycvdk.
Keckler, S. W., Dally, W. J., Khailany, B., Garland, M., & Glasco, D. (2011). GPUs and the future of parallel computing. *IEEE Micro*, *31*(5), 7–17.
Li, C. H. (2016). The performance of ML, DWLS, and ULS estimation with robust corrections in structural equation models with ordinal variables. *Psychological Methods*, *21*(3), 369–387.
Liou, M., & Yu, L. C. (1991). Assessing statistical accuracy in ability estimation: A bootstrap approach. *Psychometrika*, *56*(1), 55–67.
Loossens, T., Meers, K., Vanhasbroeck, N., Anarat, N., Verdonck, S., & Tuerlinckx, F. (2021). Efficient estimation of bounded gradient-drift diffusion models for affect on CPU and GPU. *Behavior Research Methods*.
McCulloch, C. E. (1982). Symmetric matrix derivatives with applications. *Journal of the American Statistical Association*, *77*(379), 679–682.
Meng, X. L., & Schilling, S. (1996). Fitting full-information item factor models and an empirical investigation of bridge sampling. *Journal of the American Statistical Association*, *91*(435), 1254–1267.
Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement*, *16*(2), 159–176.
Muthén, B., & Asparouhov, T. (2012). Bayesian structural equation modeling: A more flexible representation of substantive theory. *Psychological Methods*, *17*(3), 313–335.
Muthén, L. K. & Muthén, B. O. (1998–2017). *Mplus User's Guide*, (8th ed.). Los Angeles: Muthén & Muthén.
Nickolls, J., & Dally, W. J. (2010). The GPU computing era. *IEEE Micro*, *30*(2), 56–69.
NVIDIA, Vingelmann, P., & Fitzek, F. H. (2020). CUDA, release: 10.2.89. https://developer.nvidia.com/cuda-toolkit.
Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., & Chanan, G. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., & Garnett, R. (Eds.) *Advances in neural information processing systems*, (Vol. 32, pp. 8024–8035): Curran Associates Inc.
Patton, J. M., Cheng, Y., Yuan, K. H., & Diao, Q. (2014). Bootstrap standard errors for maximum likelihood ability estimates when item parameters are unknown. *Educational and Psychological Measurement*, *74*(4), 697–712.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Polyak, B. T., & Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, *30*(4), 838–855.

Reddi, S. J., Kale, S., & Kumar, S. (2018). On the convergence of Adam and beyond. In *International conference on learning representations*.

Rubin, D. B. (1976). Inference and missing data. *Biometrika*, *63*(3), 581–592.

Ruppert, D. (1988). Efficient estimations from a slowly convergent Robbins–Monro process. Technical report, Cornell University Operations Research and Industrial Engineering.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika*, *34*(4), 1–97.

Schilling, S. G., & Bock, R. D. (2005). High-dimensional maximum likelihood item factor analysis by adaptive quadrature. *Psychometrika*, *70*(3), 533–555.

Sheng, Y., Welling, W. S., & Zhu, M. M. (2014). A GPU-based Gibbs sampler for a unidimensional IRT model. *International Scholarly Research Notices*, Article ID 368149.

Sheng, Y., Welling, W. S., & Zhu, M. M. (2015). GPU-accelerated computing with Gibbs sampler for the 2PNO IRT model. In van der Ark, L. A., Bolt, D. M., Wang, WC., Douglas, J. A., & Chow, S. M. (Eds.) *Quantitative psychology research*, (pp. 59–73). Cham: Springer International Publishing.

Song, X. Y., & Lee, S. Y. (2005). A multivariate probit latent variable model for analyzing dichotomous responses. *Statistica Sinica*, 645–664.

Stevens, S. S. (1946). On the theory of scales of measurement. *Science*, *103*(2684), 677–680.

Swaminathan, H., Hambleton, R. K., & Rogers, H. J. (2006). 21 assessing the fit of item response theory models. In Rao, C., & Sinharay, S. (Eds.) *Psychometrics*, (Vol. 26, pp. 683–718).

Takane, Y., & de Leeuw, J. (1987). On the relationship between item response theory and factor analysis of discretized variables. *Psychometrika*, *52*(3), 393–408.

The NumPy Community (2021). Broadcasting. https://numpy.org/devdocs/user/basics.broadcasting.html.

Urban, C. J., & Bauer, D. J. (2021). A deep learning algorithm for high-dimensional exploratory item factor analysis. *Psychometrika*, *86*(1), 1–29.

Verdonck, S., Meers, K., & Tuerlinckx, F. (2016). Efficient simulation of diffusion-based choice RT models on CPU and GPU. *Behavior Research Methods*, *48*, 13–27.

von Davier, M. (2017). New results on an improved parallel EM algorithm for estimating generalized latent variable models. In van der Ark, L. A., Wiberg, M., Culpepper, S. A., Douglas, J. A., & Wang, W. C. (Eds.) *Quantitative psychology*, (pp. 1–8). Cham: Springer International Publishing.

van der Linden, W. J. (2016). *Handbook of item response theory, volume one: Models*, (1st ed.). London: Chapman and Hall/CRC.

Vihola, M. (2012). Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing*, *22*(5), 997–1008.

Wirth, R. J., & Edwards, M. (2007). Item factor analysis: Current approaches and future directions. *Psychological Methods*, *12*(1), 58–79.

Wu, M., Davis, R. L., Domingue, B. W., Piech, C., & Goodman, N. (2020). Variational item response theory: Fast, accurate, and expressive. arXiv:2002.00276.

Yates, A. (1987). *Multivariate exploratory data analysis: A perspective on exploratory factor analysis*. Albany: State University of New York Press.

Yuan, K. H., Cheng, Y., & Patton, J. (2014). Information matrices and standard errors for MLEs of item parameters in IRT. *Psychometrika*, *79*(2), 232–254.

Zhang, S., & Chen, Y. (2021). *Computation for latent variable model estimation: A unified stochastic proximal framework*.

Zhang, S., Chen, Y., & Liu, Y. (2020). An improved stochastic EM algorithm for large-scale full-information item factor analysis. *British Journal of Mathematical and Statistical Psychology*, *73*(1), 44–71.