Using Hamiltonian Monte Carlo to estimate the log-linear cognitive diagnosis model via Stan

Zhehan Jiang¹ · Richard Carter²

Published online: 12 June 2018
© Psychonomic Society, Inc. 2018

Abstract

The Bayesian literature has shown that the Hamiltonian Monte Carlo (HMC) algorithm is powerful and efficient for statistical model estimation, especially for complicated models. Stan, a software program built upon HMC, has been introduced as a means of psychometric modeling estimation. However, there are no systemic guidelines for implementing Stan with the log-linear cognitive diagnosis model (LCDM), which is the saturated version of many cognitive diagnostic model (CDM) variants. This article bridges the gap between Stan application and Bayesian LCDM estimation: Both the modeling procedures and Stan code are demonstrated in detail, such that this strategy can be extended to other CDMs straightforwardly.

Keywords Markov chain Monte Carlo (MCMC) · Bayesian · Cognitive diagnostic model · LCDM · Stan · Hamiltonian Monte Carlo (HMC)

The popularity of Bayesian inference has gained attention in the past decade. The number of publications indexed by Google Scholar with terms “Bayes” and/or “Bayesian” has increased to above 25% from 5% in the early 2000s (a similar finding can be found in M. D. Lee & Wagenmakers, 2014, p. 7). The core of Bayesian inferences, Markov chain Monte Carlo (MCMC) techniques can provide accurate estimation for models with high complexity, where many traditional methods cannot (see, e.g., Muthén & Asparouhov, 2012). On the other hand, the availability of software programs like WinBUGS (Lunn, Thomas, Best, & Spiegelhalter, 2000), JAGS (Plummer, 2003), and Stan (Carpenter et al., 2017) have improved the usability of Bayesian estimation in practice. Among those software programs, the Gibbs sampling (Geman & Geman, 1984), the Metropolis algorithm (MH; Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953), and other hybrid variants such as Metropolis Hastings-within-Gibbs (Gilks, 1998) are widely known. These programs are incorporated in WinBUGS and JAGS. In the current context, examples using

MCMC algorithms to estimate the parameters of several simplified cognitive diagnostic models via MH or Gibbs sampling methods can be found in Culpepper (2015), de la Torre (2009), de la Torre and Douglas (2004), and Junker and Sijtsma (2001). In addition, DeCarlo (2012) uses Bayesian approach to recognize uncertainty in DINA Q-matrix.

These algorithms have a high chance of failing convergence due to their inefficiencies in the MCMC processes, especially models with many correlated parameters, and models lacking posterior conjugacy. Sorensen, Hohenstein, and Vasishth (2016) claim that “a model with 14 fixed effects predictors and two crossed random effects by subject and item, each involving a 14×14 variance-covariance matrix” cannot be fit in WinBUGS nor JAGS. In contrast, Stan (Carpenter et al., 2017) is capable to fit the aforementioned model and other arbitrarily complex models, as it possesses the no-U-turn sampler (Hoffman & Gelman, 2014), an extension to Hamiltonian Monte Carlo (HMC; Neal, 2011) algorithm. Some simulation studies have shown that HMC is more efficient than its counterparts such as MH algorithm and the Gibbs samplers in many situations (Girolami & Calderhead, 2011). On the other hand, simulation studies conducted by Almond (2014) show that in a simple model JAGS can converge in a shorter time, despite Stan provides more effective sample sizes. Da Silva, de Oliveira, von Davier, and Bazán (2017) claim that a tailored Gibbs sampler, incorporated in the R package **dina** (Culpepper, 2015) can be more efficient than

---

✉ Zhehan Jiang
zjiang17@ua.edu

¹ University of Alabama, 309E LB Gorgas Library,
Tuscaloosa, AL 35487, USA

² University of Wyoming, Laramie, WY, USA

HMC in Stan, where OpenBUGS is less efficient. Note that the efficiency comparison result can vary from model to model and/or condition to condition. Therefore, a general statement can't conclude at this point.

HMC is the default choice in Stan software program, which has been introduced to psychometric modeling; for example, Luo and Jiao (2017) provide a tutorial on how to apply Stan for item response models (IRT). In yet another tutorial, Annis, Miller, and Palmeri (2017) delineate the steps of specifying customized distributions in Stan via linear ballistic accumulator models. What's more, Sorensen, Hohenstein, and Vasishth (2016) illustrate the procedures of loading two-level hierarchical linear models on Stan. Finally, Merkle and Wang (2018) implement Stan to structural equation models. With respect to the family of cognitive diagnosis models (CDMs), S. T. Lee (2016) present the details on specifying Stan code for the deterministic inputs, noisy “and” gate (DINA) model (Haertel, 1989; Junker & Sijtsma, 2001; Macready & Dayton, 1977), and similar work can be found in a recent contribution by da Silva, de Oliveira, von Davier, and Bazan (2017). To date, no instructions are found for estimating other advanced CDM variants via Stan. Zhan (2017) demonstrates JAGS code to fit some common CDMs including DINA, the deterministic input, noisy “or” gate (DINO) model (Templin & Henson, 2006), the linear logistic model (LLM; Maris, 1999), and finally, the saturated version of these models: the log-linear CDM (LCDM; Henson, Templin, & Willse, 2009). Although Zhan specifies Bayesian LCDM via JAGS comprehensively, the algorithm is not based upon HMC.

The LCDM has gained emphases as it provides more information than other CDM variants. In addition, it is relatively robust when the hierarchy of structure model is misspecified (Templin & Bradshaw, 2014). Note that the LCDM is essentially equivalent to the generalized DINA model (GDINA; de la Torre, 2011). For consistency purpose, the abbreviation “LCDM” is used to represent the saturated model in the following sections. Essentially, the present article aligns the LCDM estimation with HMC via Stan program. Readers are assumed to have fundamental knowledge on Bayesian inferences and statistical programming languages (see Sorensen et al., 2016, for the prerequisite for understanding the present article). The LCDM will be briefly revisited in the coming section.

The log-linear cognitive diagnostic model

Different from item response theory (IRT) that assumes a continuum for latent variables of interest, cognitive diagnostic models (CDMs; DiBello & Stout, 2007) have been developed to evaluate the strengths and weaknesses in a preverified content domain by identifying the presence or absence of multiple fine-grained attributes (or skills). The features of CDMs naturally sets latent variables of interest onto a binary scale such

that “mastery” or “nonmastery” of the attributes can be determined. As a result, CDMs can yield an attribute profile of each respondent that essentially serves as a diagnosis report. Similar to IRT, CDMs possess high interpretability for attribute structure that can be used to support or verify a certain theory, but CDMs have higher reliability and can offer richer diagnostic information to aid decision-making (Rupp & Templin, 2008). Famous CDM variants include the aforementioned DINA, as well as DINO, with noisy input; the deterministic “and” gate model (NIDA; Junker & Sijtsma, 2001); and the reparameterized unified model (RUM; Hartz, 2002). Subsequent advances in model development have produced general diagnostic models. Examples of these include the generalized deterministic input, noisy “and” gate model (G-DINA; de la Torre, 2011), general diagnostic model (GDM; von Davier, 2005), and the aforementioned LCDM. An LCDM (G-DINA with a logit link) provides great flexibility such as (1) subsuming most latent attributes, (2) enabling both additive and non-additive relationships between attributes and items simultaneously, and (3) syncing with other psychometric models, increasing insightfulness. Given these advantages, this article extends the DINA-HMC work by da Silva et al. (2017) to a more applicable situation via an LCDM strategy.

To sum, CDMs are a class of probabilistic, confirmatory, multidimensional latent-class models (von Davier, 2009). The LCDM is distinct from other CDM variants in terms of its parameterization of the measurement component. We start with the formulas for a general latent class model and move on to the specific parameterization of the LCDM. As a member of latent class models, an LCDM is mathematically defined as:

$$P(\mathbf{Y}_p = \mathbf{y}_p) = \sum_{c=1}^C v_c \prod_{i=1}^I \pi_{ci}^{y_{pi}} (1 - \pi_{ci})^{1 - y_{pi}}, \quad (1)$$

where $\mathbf{y}_p = (y_{p1}, y_{p2}, \dots, y_{pN})$ is the correct/incorrect response vector of respondent p on a test comprised of I items, and element y_{pi} is the corresponding response on item i . v_c is the probability of membership in latent class c , and π_{pi} is the probability of correct response to item i by respondent p in the class. Extended from Eq. 1, the log-likelihood function for a random sample of size N can be expressed:

$$L = \sum_{p=1}^N \log \left\{ \sum_{c=1}^C \left(v_c \prod_{i=1}^I \pi_{ci}^{y_{pi}} (1 - \pi_{ci})^{1 - y_{pi}} \right) \right\}. \quad (2)$$

To simplify computational efforts, Eq. 2 is often rewritten as:

$$L = \sum_{p=1}^N \log \left\{ \sum_{c=1}^C \left(\exp \left(\log(v_c) + \log \left(\prod_{i=1}^I \pi_{ci}^{y_{pi}} (1 - \pi_{ci})^{1 - y_{pi}} \right) \right) \right) \right\}, \quad (3)$$

where $\log \left(\prod_{i=1}^I \pi_{ci}^{y_{pi}} (1 - \pi_{ci})^{1 - y_{pi}} \right)$ can be further converted to

$$\sum_{i=1}^I \log \left(\pi_{ci}^{y_{pi}} (1 - \pi_{ci})^{1-y_{pi}} \right).$$

Suppose there are A attributes. The cognitive state of a respondent is denoted by attribute vector $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_A)$, where each element in α is a 1/0 binary variable indicating whether a respondent has mastered a th attribute: α_a . There are a total number of 2^A possible attribute patterns (i.e., classes). To illustrate, a respondent p with a pattern $\alpha = (0, 1, 1, 0)$ has mastered the second and the third attributes, but not the first and the forth ones. Similarly, if the pattern becomes $\alpha = (1, 1, 1, 1)$, it means the respondent has mastered all attributes. To identify attributes that are required to solve each item, content experts provide a Q-matrix of size $I * A$, where I and A are the numbers of items and attributes in a test, respectively. The (i, a) entry of the Q-matrix q_{ia} is 1 when item i is associated with attribute a , and otherwise $q_{ia} = 0$. Given respondent p 's attribute pattern is α_c , the conditional probability of item i can be stated as

$$\pi_{ci} = P(y_{pi} = 1 | \alpha_c) = \frac{\exp(\lambda_{i,0} + \lambda_i^T h(\alpha_c, q_i))}{1 + \exp(\lambda_{i,0} + \lambda_i^T h(\alpha_c, q_i))}, \quad (4)$$

where q_i is the set of Q-matrix entries for item i , $\lambda_{i,0}$ is the intercept parameter, where λ_i represents a vector of size $(2^A - 1) * 1$ that contains main effect and interaction effect parameters of item i , and $h(\alpha_c, q_i)$ is a vector of size $(2^A - 1) * 1$ with linear combinations of the α_c and q_i . Particularly, $\lambda_i^T h(\alpha_c, q_i)$ inside the exponent function can be expressed as

$$\lambda_i^T h(\alpha_c, q_i) = \sum_{a=1}^A \lambda_{i,1,(a)} \alpha_{ca} q_{ia} + \sum_{a=1}^A \sum_{a' > 1}^A \lambda_{i,2,(a,a')} \alpha_{ca} \alpha_{ca'} q_{ia} q_{ia'} + \dots, \quad (5)$$

where $\lambda_{i,1,(a)}$ and $\lambda_{i,2,(a,a')}$ are the main effect for a th attribute α_a and a two-way interaction effect for α_a and $\alpha_{a'}$. Since elements of α_c and q_i are binary, $h(\alpha_c, q_i)$ contains binary elements, which indicate effects needed to be estimated. For an item measuring n attributes, n -way interaction effects should be specified in $h(\alpha_c, q_i)$. Table 1 shows a concrete example of a measure with three attributes: Item 1, which measures α_1 , only has two estimates, whereas Item 3, measuring all three attributes, has eight estimates in total. The item parameters, however, do require monotonicity constraints; otherwise, the LCDM estimation is likely to encounter label-switching problems (Lao & Templin, 2016). Particularly, the expectation maximization (EM; Dempster, Laird, & Rubin, 1977) algorithm—a dominant method in CDM estimations—could produce results leading to unreasonable interpretations of item parameters, as well as disruption of the converging process. Rupp, Templin, and Henson (2010) outlined the parameter constraint approach, for example, ensuring the

positiveness of $\lambda_{i,1}$ in Eq. 5 and forcing the two-way interaction effect $\lambda_{i,2,(a,a')}$ to be bigger than the corresponding negative main effects $-\lambda_{i,1,(a)}$ and $-\lambda_{i,1,(a')}$. The evidence suggests that the parameter constraint approach would keep label consistency (Lao & Templin, 2016); however, the constrained true sampling space remains unknown, due to mathematical complexity. Note that the use of monotonicity constraints introduced bias in the estimates for parameters with small values (in the population), and therefore the constraints remain controversial in the literature. After introducing the measurement component, we then describe the structural component.

In addition to item parameters, researchers would also focus on the class membership of respondents. Given a response vector y_p , let $P(C = c | Y_p = y_p)$ as the posterior class probability for a respondent, where again, subscripts i , c , and p represent the item, latent class, and person, respectively. Based upon the EM deriving process (Dempster et al., 1977), it can be proved that

$$P(C = c | Y_p = y_p) = \frac{v_c \prod_{i=1}^I \pi_{pi} (1 - \pi_{pi})^{1-y_{pi}}}{\sum_{c=1}^{N_c} v_c \prod_{i=1}^I \pi_{pi} (1 - \pi_{pi})^{1-y_{pi}}}. \quad (6)$$

This estimate provides diagnostic information for each individual. The deriving details can be found in Knott and Bartholomew (1999, pp. 90–92). The posterior probabilities of mastery of attribute a can be further derived simply by summing all $\sum_{c'} P(C = c' | Y_p = y_p)$, where c' is any latent class vector with a th element equals 1. Note that Eqs. 1–6 are the foundations in constructing Stan code: the likelihood function is comprised of samplers that align with the equations.

Hamiltonian Monte Carlo and no-U-turn sampler

Under the umbrella of MCMC methods, HMC replaces probability distributions of the Markov chain with Hamiltonian dynamics so that the process of exploiting the target distribution can be achieved more efficiently. That is, HMC extends MH algorithm by providing more precise proposal values using Hamiltonian dynamics. In each iteration of the algorithm, the values of parameters are said to “leapfrog” to states closer to their posterior densities, short-cutting the time the MH algorithm takes by avoiding proposal values that are ultimately rejected. Once new values are proposed, the HMC algorithm uses MH methods to accept/reject choices. Therefore, compared with MH algorithms, the HMC

Table 1 Formula expression example of a log-linear cognitive diagnosis model

Item	α_1	α_2	α_3	Complete $\lambda_{i,0} + \lambda_i^T h(\alpha_c, q_i)$	Expression	Simplified Expression
1	1	0	0	$\lambda_{1,0} + \lambda_{1,1}(1) + \lambda_{1,2}(0) + \lambda_{1,3}(0) + \lambda_{1,12}(1 * 0) + \lambda_{1,13}(1 * 0) + \lambda_{1,23}(0 * 0) + \lambda_{1,123}(1 * 0 * 0)$		$\lambda_{1,0} + \lambda_{1,1}(1)$
2	0	1	1	$\lambda_{2,0} + \lambda_{2,1}(0) + \lambda_{2,2}(1) + \lambda_{2,3}(1) + \lambda_{2,12}(0 * 1) + \lambda_{2,13}(0 * 1) + \lambda_{2,23}(1 * 1) + \lambda_{2,123}(0 * 1 * 1)$		$\lambda_{2,0} + \lambda_{2,2}(1) + \lambda_{2,3}(1) + \lambda_{2,23}(1)$
3	1	1	1	$\lambda_{3,0} + \lambda_{3,1}(1) + \lambda_{3,2}(1) + \lambda_{3,3}(1) + \lambda_{3,12}(1 * 1) + \lambda_{3,13}(1 * 1) + \lambda_{3,23}(1 * 1) + \lambda_{3,123}(1 * 1 * 1)$		$\lambda_{3,0} + \lambda_{3,1}(1) + \lambda_{3,2}(1) + \lambda_{3,3}(1) + \lambda_{3,12}(1) + \lambda_{3,13}(1) + \lambda_{3,23}(1) + \lambda_{3,123}(1)$

algorithm leads to a more efficient Monte Carlo sampler (Neal, 2011).

The original version of HMC has two parameters: the trajectory length and the step size. To be specific, the step size represents the sizes between candidate solution points on a single leap trajectory, while the trajectory length defines how long a leap trajectory should be. However, Hoffman and Gelman (2014) prove that the HMC performance highly depends on the two parameters which lead to a risk of poor estimation results. For example, while larger step sizes accelerate calculations, this can have a negative cost of rejecting appropriate MCMC samples. Similarly, if the trajectory is not sufficiently long, the random walk of HMC can be inefficient. On the other hand, computation powers will be wasted if the trajectory is redundantly long. The no-U-turn sampler is proposed to reduce the HMC dependency on the two parameters (Hoffman & Gelman, 2014). Instead of using a fixed step size, the no-U-turn sampler first pre-explores the sampling space and tunes the step size to a target acceptance rate (default of 0.8) that has been proved to be optimal (Betancourt, Byrne, & Girolami, 2014). Then the no-U-turn sampler adopts a recursive tree-building algorithm that keeps doubling the leap trajectory until a U-turn is encountered; the U-turn is a sign for computationally unworthy exploration. Therefore, the trajectory length is adaptive to a certain optimal range. Technical details can be found in Hoffman and Gelman (2014) and Betancourt, Byrne, and Girolami (2014).

Stan software program

Why Stan? The primary reason is that it possesses the no-U-turn HMC sampler, which has been proved to be efficient. In addition to HMC, Stan provides other estimation options such as variational inference and penalized maximum likelihood. The second reason is that, among many software programs of its kind, Stan has, arguably, the most straightforward syntax for modeling, as demonstrated in the later sessions; users having experience in BUGS code tend to find easy to switch to Stan. The third reason is that Stan allows users to insert customized functions that compiled in C++ resulting in more flexible statistical modeling constructions and faster model estimations. The fourth reason is that the Stan community

has been contributing to the Stan improvement since its debut. The Stan functions and flexibility have gained responsive updates on an open source platform. For example, a LKJ distribution (Lewandowski, Kurowicka, & Joe, 2009)—a recent developed uniform prior on correlation matrices—is a built-in choice in Stan. Last but not least, interfacing with many mainstream programming environments such as R, Python, Stat, and Matlab easily, Stan has great accessibility. Throughout the article, the **rstan** interface in the R software (R Core Team, 2018) is implemented to call the Stan functions as well as to simulate data responses. Particularly, the interface is available by loading the **rstan** (Stan Development Team, 2016a) package on R. See <http://mc-stan.org/users/interfaces/rstan> for details about installing the **rstan**. Implementing Stan on other environments follows a similar fashion.

Follow the article structure of Luo and Jiao (2017) and Jiang and Skorupski (2017), this paragraph introduces the components of a Stan program via R. The R code consists of two parts: (1) *.stan* file specification and (2) syntax for executing R. The *.stan* file is made of six Stan code blocks, as can be seen in Table 2; of these blocks, **data**, **parameters**, and **model** are mandatory, whereas the remaining blocks are need-based. There are a few general rules being applied to each code block of the *.stan* file: (1) the symbol `//` is used to place comment, (2) the data type needs to be specified for both the data records and model variables (including those before transformations), (3) the value range constraint can be placed to meet the requirement of permissible sampling numerical space, and (4) the symbol `;` should be put at the end of each code line but before the symbol `//`. After saving a *.stan* file, R syntax is executed to call the **rstan** package, compile models, push data records into the analysis, save estimation results, and finally produce outcomes. In the following section, the aforementioned work flows, including setting arguments within the functions, will be explained step by step.

Stan code for the LCDM

To explain how Stan code should be created, assume there is a dataset containing 2,000 respondents ($N = 2,000$). Note that the parameter recovery and estimate accuracy are not discussed here; this investigation will be preserved in the later simulation study session. This dataset was based on the Q-

Table 2 Six code blocks and their definitions in a Stan program

Stan Code Block	Code Block Function
data	declare the data to be pushed into the Stan program
transformed data	transform the data passed in above
parameters	define the unknowns to be estimated and corresponding constraints
transformed parameters	transform the parameters passed in above
model	Specify prior distributions and likelihood functions
generated quantities	Generate outputs from the model such as posterior predictions

matrix listed in Table 3. The classes were labeled as “000,” “001,” “010,” “011,” “100,” “101,” “110,” and “111.” The labels indicate the mastery of attributes in alphabetical order—for example, “010” means that only the second attribute is mastered. Furthermore, following the practice shown in Table 1, $\lambda_{i,0} + \lambda_i^T h(\alpha_c, q_i)$, or namely the kernel function, for each class is outlined in Fig. 1.

The expression rules in Fig. 1 follow the conventions proposed by Rupp, Templin, and Hensen (2010, p. 206): That is, (1) I simply represents λ ; (2) the number before the symbol $_$ indicates the item number; (3) the first number after the symbol $_$ represents the item effect name, where 0, 1, and n are the labels of the intercept, main effect, and n -way interaction effects, respectively; and (4) the remaining numbers identify items that contain an attribute interaction, if there is any. To illustrate, in the last cell of Fig. 1, $I9\_0$ represents the intercept of Item 9, and $I9\_213$ represents the two-way interaction effects between the first and the third attributes. According to Rupp, Templin, and Henson (2010), in addition to ensuring the nonnegativity of the main effects, the following constraints are also required: (1) $I9\_213 > -I9\_13$ and $I9\_213 > -I9\_11$, (2) $I8\_223 > -I8\_13$ and $I8\_223 > -I8\_12$, and (3) $I7\_212 > -I7\_12$ and $I7\_212 > -I7\_11$.

In total, there are four steps for specifying the syntax: (1) the **data** and **parameters** blocks, (2) the **transformed parameters** block, (3) the **model** block, and (4) the **generated quantities** block. Figure 2 shows the specifications for the

data and **parameters** blocks in the *.stan* file. The **data** block defines the data records (inputs) that Stan absorbs into the algorithm. Since N , I , C , A , and O are all integers, *int* is specified correspondingly. Note that O is the total number of observations (i.e., responses across items and respondents). $\mathbf{Y}$ is the generated dataset in a long format such that each row contains a response, the respondent index, and the item index. For example, the first row is the observation of the first respondent at the first item, the second row is that of the first respondent at the second item, and the rest of rows follow this pattern. Alternatively, one can specify it into three separate vectors and therefore loop through the response vector, the item ID vector, and the respondent ID vector. **Alpha** is the attribute pattern matching each class, as demonstrated in Table 4; these are defined as matrices. In the **parameters** block, Line 11 specifies the class membership probability $\mathbf{v}_c = [v_1, \dots, v_C]$ as *simplex* instead of *vector*. According to the Stan user manual (Stan Development Team, 2016b), *simplex* is used to map $(K - 1)$ unconstrained variables to a K -simplex. Since the sum of $[v_1, \dots, v_C]$ is constrained to 1 at a mixture model, *simplex* was a reasonable choice to specify $\mathbf{v}_c$. Lines 12 to 35 define all item parameters as real numbers; however, $<lower = 0>$ is added to the main effects, such that their numerical spaces are nonnegative. Adding the nonnegativity constraint can effectively avoid the occurrence of label switching, which is a common problem in mixture model estimations (Stephens, 2000). This practice essentially sets truncations on the priors, and therefore the posteriors. More constraints were placed in the **model** block.

Figure 3 shows the specification for the **transformed parameters** block. Note that a few lines are skipped due to space limitations. *Plmat*—an $I * C$ matrix—is created to hold transformed parameters that follow the structure in Fig. 1. Essentially, *Plmat* is the result of applying inverse-logit transformation on the kernel functions in Fig. 1. For example, the terms inside the *inv_logit* expressions in Fig. 3 are identical to those in each cell in Fig. 1. Matching Eq. 4, *inv_logit*(x) is a Stan function for yielding $\frac{\exp(x)}{1 + \exp(x)}$, and therefore *Plmat* is essentially the π matrix.

The **model** block is specified in Fig. 4. Both *contributionsC* and *contributionsI* are temporary vectors for

Table 3 A Q-matrix for responses generation

Item No.	Attribute1	Attribute2	Attribute3
1	1	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	0	0	1
6	0	0	1
7	1	1	0
8	0	1	1
9	1	0	1

Item No	000	001	010	011	100	101	110	111
1	$l1\_0$	$l1\_0$	$l1\_0$	$l1\_0$	$l1\_0+l1\_11$	$l1\_0+l1\_11$	$l1\_0+l1\_11$	$l1\_0+l1\_11$
2	$l2\_0$	$l2\_0$	$l2\_0$	$l2\_0$	$l2\_0+l2\_11$	$l2\_0+l2\_11$	$l2\_0+l2\_11$	$l2\_0+l2\_11$
3	$l3\_0$	$l3\_0$	$l3\_0+l3\_12$	$l3\_0+l3\_12$	$l3\_0$	$l3\_0$	$l3\_0+l3\_12$	$l3\_0+l3\_12$
4	$l4\_0$	$l4\_0$	$l4\_0+l4\_12$	$l4\_0+l4\_12$	$l4\_0$	$l4\_0$	$l4\_0+l4\_12$	$l4\_0+l4\_12$
5	$l5\_0$	$l5\_0+l5\_13$	$l5\_0$	$l5\_0+l5\_13$	$l5\_0$	$l5\_0+l5\_13$	$l5\_0$	$l5\_0+l5\_13$
6	$l6\_0$	$l6\_0+l6\_13$	$l6\_0$	$l6\_0+l6\_13$	$l6\_0$	$l6\_0+l6\_13$	$l6\_0$	$l6\_0+l6\_13$
7	$l7\_0$	$l7\_0$	$l7\_0+l7\_12$	$l7\_0+l7\_12$	$l7\_0+l7\_11$	$l7\_0+l7\_11$	$l7\_0+l7\_11+l7\_12+l7\_212$	$l7\_0+l7\_11+l7\_12+l7\_212$
8	$l8\_0$	$l8\_0+l8\_13$	$l8\_0+l8\_12$	$l8\_0+l8\_12+l8\_13+l8\_223$	$l8\_0$	$l8\_0+l8\_13$	$l8\_0+l8\_12$	$l8\_0+l8\_12+l8\_13+l8\_223$
9	$l9\_0$	$l9\_0+l9\_13$	$l9\_0$	$l9\_0+l9\_13$	$l9\_0+l9\_11$	$l9\_0+l9\_11+l9\_13+l9\_213$	$l9\_0+l9\_11$	$l9\_0+l9\_11+l9\_13+l9\_213$

Fig. 1 Kernel function expressions

cumulating log-likelihood: The two temporary vectors are created for the looping process of HMC. Lines 5 to 29 specify the prior distributions for all item parameters. Note that the priors are relatively uninformative (diffused), since the mean and standard deviation of the normal distribution are 0 and 15,

```

1 data{
2   int N; // number of respondents
3   int I; // number of items
4   int C; // number of classes
5   int A; // number of attributes
6   int O; // number of observations
7   vector[3] Y[O]; // vector of response&index
8   matrix[C,A] Alpha; // attribute profile matrix
9 }
10 parameters{
11   simplex[C] Vc; // class membership probability
12   real<lower=0> l1_11; // main effect of item 1
13   real<lower=0> l2_11; // main effect of item 2
14   real<lower=0> l3_12; // main effect of item 3
15   real<lower=0> l4_12; // main effect of item 4
16   real<lower=0> l5_13; // main effect of item 5
17   real<lower=0> l6_13; // main effect of item 6
18   real<lower=0> l7_11; // main effect of item 7
19   real<lower=0> l7_12; // main effect of item 7
20   real<lower=0> l8_12; // main effect of item 8
21   real<lower=0> l8_13; // main effect of item 8
22   real<lower=0> l9_11; // main effect of item 9
23   real<lower=0> l9_13; // main effect of item 9
24   real l7_212; // interaction effect of item 7
25   real l8_223; // interaction effect of item 8
26   real l9_213; // interaction effect of item 9
27   real l1_0; // intercept effect of item 1
28   real l2_0; // intercept effect of item 2
29   real l3_0; // intercept effect of item 3
30   real l4_0; // intercept effect of item 4
31   real l5_0; // intercept effect of item 5
32   real l6_0; // intercept effect of item 6
33   real l7_0; // intercept effect of item 7
34   real l8_0; // intercept effect of item 8
35   real l9_0; // intercept effect of item 9
36 }

```

Fig. 2 Stan code for the **data** and **parameters** blocks

respectively, except in Lines 18 to 20, which possess extra constraints. When no prejudgment or educated guessing is available, it is not uncommon to use uninformative distributions. Lines 18 to 20 match the aforementioned constraints; that is, (1) $l9\_213 > -l9\_13$ and $l9\_213 > -l9\_11$, (2) $l8\_223 > -l8\_13$ and $l8\_223 > -l8\_12$, and (3) $l7\_212 > -l7\_12$ and $l7\_212 > -l7\_11$. Instead of using two lines, the *fmax* function allows the prior truncation to start at the point having the larger value. For example, if $-l9\_13 > -l9\_11$, $l9\_213$ only needs to be larger than $-l9\_13$. Line 30 specifies a Dirichlet prior for class membership probability. The function *rep_vector* (*x*, *t*) forms a vector by replicating *x* *t* times. Therefore, according to Line 30, the Dirichlet prior contains nine parameters that are all equal to 1. This practice, by setting all Dirichlet parameters equal, is essentially setting a uniform prior for a multinomial distribution (Ishwaran & Zarepour, 2002). Lines 32–44 define the (log-)likelihood function, which is illustrated in Eqs. 2 and 3. Lines 33–35 construct loops for the respondent, class, and item. Refer to Eqs. 2 and 3; one needs to either multiply/add all units in order to compute the (log-)likelihood. As one finds, Lines 38 to 41 are calculating $\log(\pi_{ci}^{y_{pi}}(1-\pi_{ci})^{1-y_{pi}})$. When $y_{pi} = 1$, $\pi_{ci}^{y_{pi}}$

Table 4 An attribute pattern matrix

Class.	Attribute1	Attribute2	Attribute3
000	0	0	0
001	0	0	1
010	0	1	0
011	0	1	1
100	1	0	0
101	1	0	1
110	1	1	0
111	1	1	1

```

1 transformed parameters{
2   matrix[I, C] PImat;
3   PImat[1,1]=inv_logit(11_0);
4   PImat[2,1]=inv_logit(12_0);
5   PImat[3,1]=inv_logit(13_0);
6   PImat[4,1]=inv_logit(14_0);
7   PImat[5,1]=inv_logit(15_0);
8   PImat[6,1]=inv_logit(16_0);
9   PImat[7,1]=inv_logit(17_0);
10  PImat[8,1]=inv_logit(18_0);
11  PImat[9,1]=inv_logit(19_0);
12  PImat[1,2]=inv_logit(11_0);
13  PImat[2,2]=inv_logit(12_0);
14  PImat[3,2]=inv_logit(13_0);
15  PImat[4,2]=inv_logit(14_0);
16  PImat[5,2]=inv_logit(15_0+15_13);
17  PImat[6,2]=inv_logit(16_0+16_13);
18  ...
19  PImat[7,7]=inv_logit(17_0+17_11+17_12+17_212);
20  PImat[8,7]=inv_logit(18_0+18_12);
21  PImat[9,7]=inv_logit(19_0+19_11);
22  PImat[1,8]=inv_logit(11_0+11_11);
23  PImat[2,8]=inv_logit(12_0+12_11);
24  PImat[3,8]=inv_logit(13_0+13_12);
25  PImat[4,8]=inv_logit(14_0+14_12);
26  PImat[5,8]=inv_logit(15_0+15_13);
27  PImat[6,8]=inv_logit(16_0+16_13);
28  PImat[7,8]=inv_logit(17_0+17_11+17_12+17_212);
29  PImat[8,8]=inv_logit(18_0+18_12+18_13+18_223);
30  PImat[9,8]=inv_logit(19_0+19_11+19_13+19_213);
31 }

```

Fig. 3 Stan code for the **transformed parameters** block

$(1-\pi_{ci})^{1-y_{pi}}$ is reduced to π_{ci} ; otherwise, it becomes $1-\pi_{ci}$. This condition mechanism is defined by the *if–else* statements in Lines 38 and 40. The function *bernoulli_lpmf*($y_{pi} \mid \pi_{ci}$) provides the log of the probability mass function: $\log(\pi_{ci}^{y_{pi}}(1-\pi_{ci})^{1-y_{pi}})$. Therefore, when the item loop (*iteri*) closes, the temporary vector *contributionsI* would become $[\log(\pi_{1i}^{y_{pi}}(1-\pi_{1i})^{1-y_{pi}}), \log(\pi_{2i}^{y_{pi}}(1-\pi_{2i})^{1-y_{pi}}), \dots, \log(\pi_{Ni}^{y_{pi}}(1-\pi_{Ni})^{1-y_{pi}})]$. Line 43 corresponds to the calculation within the exponential function in Eq. 3. $\log(v_c)$ and $\sum_{i=1}^I \log(\pi_{ci}^{y_{pi}}(1-\pi_{ci})^{1-y_{pi}})$ are realized by $\log(Vc[iterc])$ and $\text{sum}(\text{contributionsI})$, respectively, within the **model** block. Accordingly, $\log(v_c) + \log\left(\prod_{i=1}^I \pi_{ci}^{y_{pi}}(1-\pi_{ci})^{1-y_{pi}}\right)$ is saved in the temporary vector *contributionsC*. Last but not least, Line 45 exponentiates all elements in the *contributionsC* vector, sums those exponentiated parameters, performs logarithm transformation, and finally adds the respondent's log-likelihood to the total log-likelihood. In other words, $+=\log\_sum\_exp(x)$ executes the core part of Eq. 3, $\sum_{p=1}^N \log\left\{\sum_{c=1}^C (\exp(x))\right\}$.

```

1 model {
2   vector[C] contributionsC;
3   vector[I] contributionsI;
4
5   //Prior
6   11_11~normal(0,15);
7   12_11~normal(0,15);
8   13_12~normal(0,15);
9   14_12~normal(0,15);
10  15_13~normal(0,15);
11  16_13~normal(0,15);
12  17_11~normal(0,15);
13  17_12~normal(0,15);
14  18_12~normal(0,15);
15  18_13~normal(0,15);
16  19_11~normal(0,15);
17  19_13~normal(0,15);
18  17_212~normal(0,15)T[fmax((-1)*17_11,(-1)*17_12),];
19  18_223~normal(0,15)T[fmax((-1)*18_12,(-1)*18_13),];
20  19_213~normal(0,15)T[fmax((-1)*19_11,(-1)*19_13),];
21  11_0~normal(0,15);
22  12_0~normal(0,15);
23  13_0~normal(0,15);
24  14_0~normal(0,15);
25  15_0~normal(0,15);
26  16_0~normal(0,15);
27  17_0~normal(0,15);
28  18_0~normal(0,15);
29  19_0~normal(0,15);
30  Vc~dirichlet(rep_vector(1.0, C));
31
32  //Likelihood
33  for (iterp in 1:N){
34    for (iterc in 1:C){
35      for (itero in 1:O){
36        if (Y[itero,2]==interp)
37          iteri = Y[itero,3]
38        if (Y[itero,1] == 1)
39          contributionsI[iteri]=bernoulli_lpmf(1|PImat[iteri,iterc]);
40        else
41          contributionsI[iteri]=bernoulli_lpmf(0|PImat[iteri,iterc]);
42      }
43      contributionsC[iterc]=log(Vc[iterc])+sum(contributionsI);
44    }
45    target+=log_sum_exp(contributionsC);
46  }

```

Fig. 4 Stan code for the **model** block

The last block is **generated quantities** (Fig. 5), which is used to produce the posterior distributions. Line 2 is the posterior probability matrix for attribute mastery that corresponds

```

1 generated quantities {
2   matrix[N,A] prob_resp_attr;
3   vector[C] prob_resp_class[N];
4   vector[C] prob_joint;
5   vector[I] contributionsI;
6   vector[C] prob_attr_class;
7
8   for (iterp in 1:N){
9     for (iterc in 1:C){
10      for (itero in 1:O){
11        if (Y[itero,2]==interp)
12          iteri = Y[itero,3]
13        if (Y[itero,1] == 1)
14          contributionsI[iteri]=bernoulli_lpmf(1|PImat[iteri,iterc]);
15        else
16          contributionsI[iteri]=bernoulli_lpmf(0|PImat[iteri,iterc]);
17      }
18      prob_joint[iterc]=(Vc[iterc])*exp(sum(contributionsI));
19    }
20    prob_resp_class[iterp]=prob_joint/sum(prob_joint);
21  }
22
23  for (iterp in 1:N){
24    for (itera in 1:A){
25      for (iterc in 1:C){
26        prob_attr_class[iterc] = prob_resp_class[iterp,iterc] * Alpha[iterc,itera];
27      }
28      prob_resp_attr[iterp,itera] = sum(prob_attr_class);
29    }
30  }
31 }

```

Fig. 5 Stan code for the **generated quantities** block

to $P(\alpha_{pc} = 1 \mid \mathbf{Y}_p = \mathbf{y}_p)$, where Line 3 defines $P(C = c \mid \mathbf{Y}_p = \mathbf{y}_p)$, such that each respondent has a C -element class probability vector. Lines 4 to 6 are once again temporary containers for the algorithm loops. Lines 8–21 are fairly similar to what is specified in Fig. 4, except that Line 16 is calculating $v_c \prod_{i=1}^I \pi_{pi}(1 - \pi_{pi})^{1 - y_{pi}}$, and Line 20 refers to Eq. 6. Equation 7 in turn can be referred to Lines 23–31, where the posterior probabilities of mastery of attributes are generated for producing posterior distributions. At the current step, the *.stan* file is completed and can be saved as *LCDM.stan*.

Stan outputs

Before analyzing data, R needs to load the **rstan** interface by calling *library(rstan)*. The *.stan* file should be compiled by executing *model <- stan_model("LCDM.stan")*. The first time a model is executed, *LCDM.stan* will take some time to compile before sampling. On subsequent runs, it will only recompile if the *.stan* file is altered. In addition, the data and other input variables need to be organized in a list and passed to Stan via syntax *data = list(Y = Y, A = A, N = N, I = I, C = C, Alpha = Alpha)*. These inputs were pre-defined prior to the analysis. Finally, the estimation task is performed by executing a command such as *stan.result <- sampling(model, data = data, iter = 8000)*. No specifying the number of iterations or the number of chains would automatically set the configurations to default values—2,000 and 4, respectively. The machine executed **rstan** is Lenovo IdeaPad with 8GB RAM and a 2.6 GHz i7 6th Gen 4core Intel processor. All four cores were implemented in this article for faster computing.

Before using the Stan results, the convergence should be confirmed via the Gelman–Rubin convergence diagnostic statistics (represented by *rhat* in the **rstan** output; Gelman & Rubin, 1992). Values close to 1 indicate a high chance that the multiple chains have converged to the same distribution and therefore the mixing process is sufficient. In addition, the effective sample size (ESS), an estimate of the number of independent samples from the HMC posterior distribution, is recommended to accompany *rhat* to assess convergence (Gelman, Lee, & Guo, 2015): a larger ESS implies a lower risk of autocorrelation. Finally, Stan provides posterior distributions of parameters of interest that includes point estimates, standard deviations, and quantile values.

Simulation study

Simulation design

To demonstrate the utility of estimating an LCDM via Stan, a small simulation study is provided here. Note that a

comprehensive investigation on how Stan performs in this vein of estimation is beyond the scope of this article, as Bayesian studies often involve intricate trials of priors and tuning strategies. We refer the more comprehensive study for future works. For the purpose of this study we adopt the simulation design by Templin and Bradshaw (2014). In this design, there were 3,000 respondents, 30 items, and three attributes resulting in eight classes in total. To avoid a confound potentially caused by the magnitudes of item parameters, the main effects were all set to a value of 2, the interactions were all set to a value of 1, and the intercept was set to $-0.5 \cdot \lambda_i^T \mathbf{h}(\alpha_1, \mathbf{q}_i)$, where $\alpha_1 = \{1, 1, \dots, 1\}$. According to Templin and Bradshaw, the setting of the intercept can make attribute misclassifications roughly equal. Based upon LCDM analysis of the Examination for the Certificate of Proficiency in English (ECPE), the class membership probability was set to [0.301, 0.129, 0.012, 0.175, 0.009, 0.018, 0.011, 0.346] for classes labeled as “000,” “001,” “010,” “011,” “100,” “101,” “110,” and “111”, respectively. Finally, a balanced Q-matrix with each item measuring either one or two attributes was utilized. The study was replicated for 300 times.

For reference, the **CDM** package (George, Robitzsch, Kiefer, Groß, & Ünlü, 2016) in R was used to estimate the model in addition to the **rstan**. To estimate an LCDM, the **CDM** package uses the EM algorithm tailored by de la Torre (2011). Similar estimation practices can be found in the **GDINA** package, as detailed by Ma and de la Torre (2016). In addition, the **Mplus** software (Muthén & Muthén, 2013) can also be used to estimate an LCDM (see Templin & Hoffman, 2013, for syntax instruction). Both the **CDM** and **GDINA** packages allow users to set the monotonicity constraints via a command line, while the **Mplus** requires the specification one-by-one. With respect to Stan configurations, both informative and uninformative priors for item parameters were specified. To be concrete, both prior sets use 0 as the mean for each item parameter prior, while the informative set use 5 as the standard deviation and the other set uses 15. The chain number remained default while the HMC iteration number was set to 5,000. The thin parameter was set to 5. We used the posterior mean as the estimate of a parameter.

Simulation results

A pilot study was conducted to ensure the convergence of the Bayesian estimations: ESSs and *rhats* were recorded and therefore investigated for all parameters across 20 replications. The ESSs ranged from 94 to 200 out of 200 samples (i.e., the effective ratios were between .47 to 1.00). In addition, the *rhats* ranged from 0.99 to 1.02. This pilot study validated that 1,000 iterations were sufficient for reaching a convergence. Therefore, the simulation results could be considered trustworthy. Besides, the average elapsed time, including the

warm-up and the sampling, is 43 min for four chains. Note that the computation time is expected to vary substantively when (1) the number of iterations, (2) the number of chains, (3) the selection of priors, (4) the complexity of models, and (5) the configurations of machines are different.

Two generic outcomes are presented here: (1) parameter recovery measured by bias and root mean squared error (RMSE), and (2) classification accuracy measured by the attribute profile classification rate and marginal attribute classification rate. In particular, the attribute profile classification rate is the proportion of respondents that were correctly classified as having a given class (namely attribute pattern), while the marginal attribute classification rate is the proportion of times a single attribute was classified correctly, across all respondents and attributes.

Figure 6 presents the relative biases and RMSEs in box plots that arrange the results in terms of item parameter categories: intercept, main effect, and interaction effect. The dark line inside the box represents the median. The top of box is 75th percentile, and bottom of box is 25th percentile. The end points of the lines (i.e., whiskers) are at a distance of $1.5 \times \text{IQR}$, where the interquartile range, or IQR, is the distance between the 25th and 75th percentiles. The points outside the whiskers are marked as dots and are normally considered as extreme points. In addition, the means and the standard deviations of the estimates of each category were appended. As the upper panel indicates, the HMC method with informative priors (labeled as *staninf*) yielded more precise results than its competitors. In particular, the HMC method, with informative priors, was about 0.2 more precise than the EM approach in terms of the relative biases of both intercept and interaction effects. All three methods produced scattered relative biases in the interaction effects. However, the HMC method with informative priors had fewer and less-extreme outliers. Note that with uninformative priors, the HMC method yielded results similar to those from the EM approach, except in the main effect category. The lower panel contains the RMSE results. The HMC method with informative priors had the lowest means for RMSEs in both intercept and interaction terms, while the EM approach has the lowest mean RMSE in main effect. It can be seen that the RMSEs tended to vary more in the HMC method with uninformative priors. Overall, most point estimates of the main effect parameters were close to the true values, some intercept point estimates deviated from their true values substantively, and finally, the point estimates of the interaction terms were less accurate. This finding is not uncommon, because in a latent-modeling context, the estimate accuracy of interaction effect and its related main effect(s) tend to be lower than the intercept effects (Jiang, Wang, & Weiss, 2016).

The class probability estimates tend to be very similar across all three approaches. The means of the relative biases (across eight classes) are 0.013, 0.016, and 0.014 for the EM approach and the HMC with uninformative and informative

priors, respectively. Correspondingly, the RMSEs are 0.018, 0.021, and 0.019 for the three approaches, following the aforementioned order. For the three attributes, the profile classification rates of the EM approach are .98, .98, and .97, while those of the HMC with uninformative priors are .97, .97, and .96, and finally those of the HMC with informative priors are .99, .97, and .97. To emphasize, the priors were varied in the item parameters only; the priors for the class probabilities were identical for the two HMC estimations. These results show that the classification qualities, from the perspectives of both class probability estimates and attribute assignments, were relatively high for all three estimations, while the EM approach seemed to perform slightly better.

Discussion and conclusion

Unlike other works of this kind, the present article uses Stan to estimate the LCDM only. To emphasize, the reason is that the LCDM is the parent model of other common CDM variants (i.e., the specification of the LCDM is more complicated). Tuning the code to other CDMs simply needs to constrain and/or transform certain parameters. For instance, allowing the intercepts and interaction effects to be estimated but setting main effects to zeros would transform an LCDM to a DINA model (see Rupp et al., 2010, pp. 159–167, for details about converting the LCDM to other CDMs). In addition, when one masters the knowledge on applying Stan to a complicated model, it is straightforward to extend the strategy to those of simpler variants. Overall, estimating an LCDM through Stan is straightforward and it can be tuned for estimating other variants by adding zero-constraints. This practice allows users to use Bayesian features for model estimations and therefore is recommended.

Despite the instruction provided in the present article functions appropriately, there may have space for specifying *.stan* file code and model configurations more efficiently. Stan code has substantive variety such that an identical model can be specified in multiple ways resulting in different computational speeds and estimates. In addition, changing the input parameters of Stan, such as draws per chain and iteration numbers, may cause differences in final results, although truly converged models should have no systemic discrepancies between replications with nonidentical algorithm settings.

Like other Bayesian techniques, the results provided Stan needs model fit assessment. There are four typical problems that an insufficient estimation may encounter: (1) lack of mixing, (2) nonstationarity, (3) autocorrelation, and (4) divergent transitions. The first problem can be handled by providing different starting values and multiple chains. As demonstrated earlier, when the Gelman–Rubin convergence diagnostic statistics approach 1, the sampling is claimed to be from a well-specified posterior. Note that a poorly specified model

Fig. 6 Box plots of simulation results

could cause the mixing failed repetitively, even adopting the multiple starting values strategy. Nonstationarity occurs when a model is misspecified and/or insufficient number of iterations; therefore, if time is not a concern, a large number of iterations is always preferred. Autocorrelation is a commonly seen issue in a weakly identified model; that is, the drawing of certain parameter(s) depends highly on the previous draws, resulting in unreliable estimates. Thinning and reparametrizing are two solutions to autocorrelation.

Arguably, thinning a Markov chain can result in a loss of information, whereas reparametrizing in many situations complicates the modeling procedures. The last problem—divergent transitions—is about insufficient sampling under irregular posterior densities. This issue can be fixed by manually increasing the desired average acceptance value close to 1 (e.g., .99). In **rstan**, this tuning is executed by specifying `stan(..., control = list(adapt_delta = 0.99), ...)`. As a trade-off, the mixing process will be slowed down.

Stan doesn't provide fit statistics such as the Akaike information criterion (AIC), the Bayesian information criterion (BIC), and the deviance information criterion (DIC). In particular, providing DIC is a standard function provided in WinBUGS, JAGS, and other Bayesian software programs. However, DIC is known to have some challenges, since it is based on a point estimate. For example, DIC can produce negative values, which are prohibited (Plummer, 2008; van der Linde, 2005). Instead of using the aforementioned statistics, Stan adopts the practice of approximating leave-one-out (LOO) cross-validation error during sampling processes. This practice generates probabilistic estimates of the degree to which the model has the highest probability to produce the best predictions. As a result, LOO can be used to serve as an information criterion (LOOIC) for model comparisons. Similar to AIC, BIC, and DIC, a smaller LOOIC means that the model generates more plausible predictions than does one with a larger LOOIC (see Vehtari, Gelman, & Gabry, 2017, for details about LOO). In addition, this approach can be further adapted for more trustworthy invariance tests (Shi et al., 2018, manuscript submitted for publication; Shi, Song, Liao, Terry, & Snyder, 2017).

The intention of this article is to act as a tutorial, and therefore the utility of HMC can be realized for LCDM estimation. The simulation provides evidence that HMC can appropriately and accurately estimate an LCDM. However, as an initial illustrative work, this article does not provide comprehensive instructions about how to improve the estimation speed and quality. In addition, it is known that different prior distributions are always an important concern in Bayesian inference. If a researcher has a scientific judgment and/or well-educated guess, this “non-data-based” information can be blended in with the estimation, as the previous example demonstrated. Alternatively, to study the robustness of a model estimation, researchers can conduct sensitivity analysis by trying different prior distributions. Although this task is beyond the scope of the present article, future studies are encouraged to focus on varying the priors and so examining the sensitivity of Bayesian LCDM. Note that incorporating prior distributions could be also realized in other algorithms, such as the EM approach, such that the ML estimation essentially becomes MAP estimation. Further research could focus on diverse situations, such as the impact of Q-matrix designs on the Bayesian estimations (Liu, 2017) and the selection of prior distributions with missing responses in Q-matrix validation (Dai, Svetina, & Chen, 2018).

References

- Almond, R. (2014). Comparison of two MCMC algorithms for hierarchical mixture models. In Bayesian Modeling Application Workshop at the Uncertainty in Artificial Intelligence Conference (pp. 1–19). Corvallis, OR: AUAI Press.
- Annis, J., Miller, B. J., & Palmeri, T. J. (2017). Bayesian inference with Stan: A tutorial on adding custom distributions. *Behavior Research Methods*, 49, 863–886. doi:<https://doi.org/10.3758/s13428-016-0746-9>
- Betancourt, M. J., Byrne, S., & Girolami, M. (2014). *Optimizing the integrator step size for Hamiltonian Monte Carlo*. arXiv:1411.6669
- Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., ... Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, 20, 1–37.
- Culpepper, S. A. (2015). Bayesian estimation of the DINA model with Gibbs sampling. *Journal of Educational and Behavioral Statistics*, 40, 454–476.
- Dai, S., Svetina, D., & Chen, C. (2018). Investigation of missing responses in Q-matrix validation. *Applied Psychological Measurement*. Advance online publication. doi:<https://doi.org/10.1177/0146621618762742>
- de la Torre, J. (2009). DINA model and parameter estimation: A didactic. *Journal of Educational and Behavioral Statistics*, 34, 115–130.
- de la Torre, J. (2011). The generalized DINA model framework. *Psychometrika*, 76, 179–199.
- de la Torre, J., & Douglas, J. A. (2004). Higher-order latent trait models for cognitive diagnosis. *Psychometrika*, 69, 333–353.
- DeCarlo, L. T. (2012). Recognizing uncertainty in the Q-matrix via a Bayesian extension of the DINA model. *Applied Psychological Measurement*, 36, 447–468.
- 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*, 39, 1–38.
- Gelman, A., Lee, D., & Guo, J. (2015). Stan: A probabilistic programming language for Bayesian inference and optimization. *Journal of Educational and Behavioral Statistics*, 40, 530–543.
- Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, 7, 457–472. doi:<https://doi.org/10.2307/2246093>
- Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6, 721–741. doi:<https://doi.org/10.1109/TPAMI.1984.4767596>
- George, A. C., Robitzsch, A., Kiefer, T., Groß, J., & Ünlü, A. (2016). The R package CDM for cognitive diagnosis models. *Journal of Statistical Software*, 74(2), 1–24.
- Gilks, W. R. (1998). Full conditional distributions. In W. R. Gilks, S. Richardson, & D. J. Spiegelhalter (Eds.), *Markov chain Monte Carlo in practice* (pp. 75–88). Boca Raton, FL: Chapman & Hall.
- Girolami, M., & Calderhead, B. (2011). Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B*, 73, 123–214.
- Haertel, E. H. (1989). Using restricted latent class models to map the skill structure of achievement items. *Journal of Educational Measurement*, 26, 301–321.
- Hartz, S. M. (2002). *A Bayesian framework for the unified model for assessing cognitive abilities: Blending theory with practicality* (Doctoral dissertation), University of Illinois at Urbana-Champaign, IL.
- Henson, R. A., Templin, J. L., & Willse, J. T. (2009). Defining a family of cognitive diagnosis models using log-linear models with latent variables. *Psychometrika*, 74, 191–210.
- Hoffman, M. D., & Gelman, A. (2014). The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15, 1593–1623.
- Ishwaran, H., & Zarepour, M. (2002). Dirichlet prior sieves in finite normal mixtures. *Statistica Sinica*, 941–963.
- Jiang, S., Wang, C., & Weiss, D. J. (2016). Sample size requirements for estimation of item parameters in the multidimensional graded

- response model. *Frontiers in Psychology*, 7, 109. doi:<https://doi.org/10.3389/fpsyg.2016.00109>
- Jiang, Z., & Skorupski, W. (2017). A Bayesian approach to estimating variance components within a multivariate generalizability theory framework. *Behavior Research Methods*. Advance online publication. doi:10.3758/s13428-017-0986-3
- Junker, B. W., & Sijtsma, K. (2001). Cognitive assessment models with few assumptions, and connections with nonparametric item response theory. *Applied Psychological Measurement*, 25, 258–272.
- Knott, M., & Bartholomew, D. J. (1999). *Latent variable models and factor analysis* (No. 7). Edward Arnold.
- Lao, H., & Templin, J. (2016, April). Estimation of diagnostic classification models without constraints: Issues with class label switching. Paper presented at Annual Meeting of the National Council on Measurement in Education, Washington, DC.
- Lee, M. D., & Wagenmakers, E. J. (2014). *Bayesian cognitive modeling: A practical course*. New York, NY: Cambridge University Press.
- Lee, S. T. (2016, November 21). DINA model with independent attributes. Retrieved from http://mc-stan.org/documentation/case-studies/dina_independent.html.
- Lewandowski, D., Kurowicka, D., & Joe, H. (2009). Generating random correlation matrices based on vines and extended onion method. *Journal of Multivariate Analysis*, 100, 1989–2001.
- Liu, R. (2017). Misspecification of attribute structure in diagnostic measurement. *Educational and Psychological Measurement*. <https://doi.org/10.1177/0013164417702458>.
- Lunn, D. J., Thomas, A., Best, N., & Spiegelhalter, D. (2000). WinBUGS—A Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing*, 10, 325–337.
- Luo, Y., & Jiao, H. (2017). Using the Stan program for Bayesian item response theory. *Educational and Psychological Measurement*, 78, 384–408. doi:<https://doi.org/10.1177/0013164417693666>
- Ma, W., & de la Torre, J. (2016). GDINA: The Generalized DINA model framework (R package version 0.13.0). Available online at <http://CRAN.R-project.org/package=GDINA>.
- Macready, G. B., & Dayton, C. M. (1977). The use of probabilistic models in the assessment of mastery. *Journal of Educational Statistics*, 2, 99–120.
- Maris, E. (1999). Estimating multiple classification latent class models. *Psychometrika*, 64, 187–212.
- Merkle, E. C., & Wang, T. (2018). Bayesian latent variable models for the analysis of experimental psychology data. *Psychonomic Bulletin & Review*, 25, 256–270. doi:<https://doi.org/10.3758/s13423-016-1016-7>
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1087–1092.
- Muthén, B., & Asparouhov, T. (2012). Bayesian structural equation modeling: A more flexible representation of substantive theory. *Psychological Methods*, 17, 313–335. doi:<https://doi.org/10.1037/a0026802>
- Muthén, L. K., & Muthén, B. O. (2013). *Mplus user's guide* (Version 7.1) [Computer software and manual]. Los Angeles, CA: Muthén & Muthén.
- Neal, R. M. (2011). MCMC using Hamiltonian dynamics. In S. Brooks (Ed.), *Handbook of Markov Chain Monte Carlo* (pp. 113–162). Boca Raton, FL: CRC Press/Taylor & Francis.
- Plummer, M. (2003). JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. Paper presented at the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria.
- Plummer, M. (2008). Penalized loss functions for Bayesian model comparison. *Biostatistics*, 9, 523–539.
- R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from www.R-project.org/
- Rupp, A. A., Templin, J., & Henson, R. A. (2010). *Diagnostic measurement: Theory, methods, and applications*. New York, NY: Guilford Press.
- Rupp, A. A., & Templin, J. L. (2008). Unique characteristics of diagnostic classification models: A comprehensive review of the current state-of-the-art. *Measurement: Interdisciplinary Research and Perspectives*, 6, 219–262.
- Shi, D., Song, H., Liao, X., Terry, R., & Snyder, L. A. (2017). Bayesian SEM for specification search problems in testing factorial invariance. *Multivariate Behavioral Research*, 52, 430–444.
- da Silva, M. A., de Oliveira, E. S. B., von Davier, A. A., & Bazán, J. L. (2017). Estimating the DINA model parameters using the No-U-Turn Sampler. *Biometrical Journal*. Advance online publication. doi:<https://doi.org/10.1002/bimj.201600225>
- Sorensen, T., Hohenstein, S., & Vasishth, S. (2016). Bayesian linear mixed models using Stan: A tutorial for psychologists, linguists, and cognitive scientists. *Quantitative Methods for Psychology*, 12, 175–200.
- Stan Development Team. (2016a). rstan: R interface to Stan (R package version 2.0.3). Retrieved from <http://mc-stan.org>
- Stan Development Team. (2016b). Stan: A C++ library for probability and sampling (Version 2.8.0). Retrieved from <http://mc-stan.org>
- Stephens, M. (2000). Dealing with label switching in mixture models. *Journal of the Royal Statistical Society: Series B*, 62, 795–809.
- Templin, J., & Bradshaw, L. (2014). Hierarchical diagnostic classification models: A family of models for estimating and testing attribute hierarchies. *Psychometrika*, 79, 317–339.
- Templin, J., & Hoffman, L. (2013). Obtaining diagnostic classification model estimates using Mplus. *Educational Measurement: Issues and Practice*, 32, 37–50.
- Templin, J. L., & Henson, R. A. (2006). Measurement of psychological disorders using cognitive diagnosis models. *Psychological Methods*, 11, 287–305. <https://doi.org/10.1037/1082-989X.11.3.287>
- van der Linde, A. (2005). DIC in variable selection. *Statistica Neerlandica*, 59, 45–56.
- Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 27, 1413–1432.
- von Davier, M. (2009). Some notes on the reinvention of latent structure models as diagnostic classification models. *Measurement: Interdisciplinary Research and Perspectives*, 7, 67–74.
- Zhan, P. (2017). Using JAGS for Bayesian cognitive diagnosis models: A tutorial. arXiv:1708.02632