Article

Using the Stan Program for Bayesian Item Response Theory

Educational and Psychological Measurement
2018, Vol. 78(3) 384–408

DOI: 10.1177/0013164417693666
journals.sagepub.com/home/epm
SAGE

Yong Luo[1] and Hong Jiao[2]

Abstract

Stan is a new Bayesian statistical software program that implements the powerful and efficient Hamiltonian Monte Carlo (HMC) algorithm. To date there is not a source that systematically provides Stan code for various item response theory (IRT) models. This article provides Stan code for three representative IRT models, including the three-parameter logistic IRT model, the graded response model, and the nominal response model. We demonstrate how IRT model comparison can be conducted with Stan and how the provided Stan code for simple IRT models can be easily extended to their multidimensional and multilevel cases.

Keywords

item response theory (IRT), Markov chain Monte Carlo (MCMC), Bayesian

Introduction

In the past two decades, Bayesian item response theory (IRT) modeling has become increasingly popular due to the advance of computing power and the Markov chain Monte Carlo (MCMC) algorithms. Multiple software programs became available to implement some MCMC algorithms, including WinBUGS (Lunn, Thomas, Best, & Spiegelhalter, 2000), OpenBUGS (Spiegelhalter, Thomas, Best, & Lunn, 2010), JAGS (Plummer, 2003), PROC MCMC in SAS, and Mcmcpack (Martin, Quinn, & Park, 2011) in R. In those software programs, the Gibbs sampling (Geman & Geman,

[1]National Center for Assessment in Higher Education, Riyadh, Saudi Arabia
[2]University of Maryland, College Park, MD, USA

Corresponding Author:
Yong Luo, National Center for Assessment in Higher Education, King Khalid Road, Riyadh, 11537, Saudi Arabia.
Email: jackyluoyong@gmail.com

1984) and the Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953) are most frequently used. Despite their popularity, these sampling algorithms have some limitations as well, especially the long computation time required for model convergence due to their inefficiency in exploring the posterior parameter space (Neal, 1993).

Stan (Carpenter et al., 2017) is a new Bayesian software program implementing the no-U-turn sampler (Hoffman & Gelman, 2014), an extension to the Hamiltonian Monte Carlo (HMC; Neal, 2011) algorithm. HMC is considerably faster than the Gibbs sampler and the Metropolis algorithm because it explores the posterior parameter space more efficiently. It does so by pairing each model parameter with a momentum variable, which determines HMC's exploration behavior of the target distribution based on the posterior density of the current drawn parameter value[1] and hence enables HMC to ''suppress the random walk behavior in the Metropolis algorithm'' (Gelman, Carlin, Stern, & Rubin, 2014, p. 300). Consequently, Stan is considerably more efficient than the traditional Bayesian software programs. As stated in the Stan User Manual (Stan Development Team, 2016a, p. 541), ''Stan might work fine with 1000 iterations with an example where BUGS would require 100,000 for good mixing.'' Another advantage of Stan is that unlike WinBUGS and OpenBUGS, it does allow the use of improper priors. In addition, it allows interface with other software programs such as R, Python, Matlab, Stata, Julia, as well as its compatibility with all three major operating platforms, namely Linux, Mac, and Windows.

Curtis (2010) introduced WinBUGS codes for several IRT models while other researchers (Ames & Samonte, 2015; Stone & Zhu, 2015) presented on using SAS PROC MCMC for IRT model parameter estimation. Despite its efficiency and easy accessibility in different interfaces, no article collectively introduces Stan codes for IRT model parameter estimation. Stan user manual (Stan Development Team, 2016a) introduces the one-parameter logistic (1PL) and two-parameter logistic (2PL) IRT models. However, if one is interested in extended IRT models, it may take a long time to learn the basics and figure out these extensions.

To promote the accessibility of Stan, this article intends to present Stan codes for three more generalized IRT models and their multidimensional and multilevel extensions. The three models are the three-parameter logistic (3PL) dichotomous IRT model, the graded response model (GRM), and the nominal response model (NRM).[2] Essentially, the 3PL model is a general case for dichotomous item responses while the other two polytomous IRT models are the difference models and the divide-by-total models, respectively (Thissen & Steinberg, 1986). Other dichotomous IRT models such as the 1PL and 2PL IRT models, and polytomous IRT models such as the partial credit model (Masters, 1982) and the rating scale model (Andrich, 1978) can be easily developed by modifying the Stan codes provided. With the introduction to the multilevel and multilevel extensions, many more available IRT models could be possible. Furthermore, this article specifically focuses on **rstan**, an R package that interfaces with Stan in the R computing environment that has become increasingly

popular among psychometricians and applied researchers (e.g., Rusch, Mair, & Hatzinger, 2013).

The rest of this article is organized into three parts. The first part describes the basic components of a Stan program and the codes for the three IRT models. The second part illustrates with both real and simulated data sets how to conduct model comparison and fit multidimensional and multilevel IRT models in Stan. This article concludes with a discussion regarding the use and limitations of Stan.

Components of a Stan Program

The main function in the **rstan** package is *stan*, which calls the Stan software program to estimate a specified statistical model. In order to use the *stan* function, a Stan program has to be specified either as a separate Stan file ending with ''stan'' as the suffix or as an object in the R environment. In the first part of this article, we follow the former approach and save a separate stan file for each IRT model in the R working directory folder; later in the second and third examples, we demonstrate the latter approach in which the Stan codes are embedded as part of an R program. We start with an introduction to the basic building blocks of a Stan program using the codes in Listings 1 to 3 in the appendix with related Listing 1 code embedded in the text. Each code also contains comments (which are preceded by ''//'' in Stan) at places where deemed appropriate.

Data Block

The first necessary component of a Stan model is the data block, in which a researcher specifies the relevant data information and the data itself. The data block, as well as any other block in a Stan file, contains a pair of curly brackets ''{ }'' in which Stan codes are written. For dichotomous IRT models, three pieces of information are usually provided in the data block: the number of examinees, the number of items, and a response matrix (Listing 1, lines 2-4) as follows:

```
1  data {
2  int<lower=0> n_student;
3  int<lower=0> n_item;
4  int<lower=0,upper=1> Y[n_student,n_item];
5  }
```

For polytomous IRT models, the number of categories for each item, which is an integer if all items have the same number of categories or a vector of integers for items with different response categories (Listing 2-3, lines 2-5), needs to be specified. One big difference between Stan and BUGS is that data type has to be specified in Stan. The number of students and the number of items are specified as *int*

(integers), while the response matrix is specified to be an array of integers, followed by its dimension. As the polytomous items in the demo codes contain four response categories, the number of categories is specified as an integer (e.g., Listing 2, line 2).

Stan provides an extra layer of data check by allowing the user to provide a data boundary in a pair of angle brackets "< >" after the specification of data type. For example, in the 3PL model, the elements in the response matrix are specified to range from 0 to 1 (Listing 1, line 4). If one element in the matrix has a value of 2, Stan will produce a warning message to notify the user that some value in the response matrix is out of range. Similarly, in a polytomous IRT model, the elements in the response matrix ranging from one to the number of categories should be specified accordingly in the data block (e.g., Listing 2, line 5).

Parameters Block

Following the data block is the parameters block in which model parameters are specified. In the context of IRT, model parameters usually include latent ability and item parameters such as difficulty, discrimination, and pseudoguessing parameters. Hyperparameters in the priors for any model parameters need to be specified in this block as well. If any model parameter or hyperparameter that appears in the subsequent model block is not specified in this block, Stan will provide warnings that a particular variable is not found and stop. Therefore, it is essential that a Stan user knows what model parameters and hyperparameters (if any) are in the model of interest and specify them accordingly and exhaustively. In the following, we provide a brief description of each of the three IRT models, followed by a discussion of how the corresponding model parameters and hyperparameters are specified.

The 3PL IRT model can be expressed as

$$p_{ij}(u_{ij}=1|\theta_i,a_j,b_j,c_j)=c_j+\frac{(1-c_j)}{1+\exp(-a_j(\theta_i-b_j))}, \quad (1)$$

where u_{ij} is the response of examinee i to item j, $\theta_i$is the latent ability of examinee i, a_j, b_j, and $c_j$are the discrimination, difficulty, and the pseudoguessing parameter or lower asymptote of item j. The 3PL model can be reduced to the 2PL IRT model by constraining c_j to be zero, and the 1PL IRT model can be obtained by further constraining a_j to be the same across all items. Furthermore, when the discrimination parameters are constrained to be 1, the Rasch model will result.

Model parameters in the 3PL IRT model that need to be specified in the parameter block include all the parameters on the right side of Equation (1) (Listing 1, lines 7-10) and their hyperparameters. In this illustration, a normal distribution with unknown mean (*mu_beta*) and unknown standard deviation (*sigma_beta*) is specified as a prior for the item difficulty parameter and a lognormal distribution with a mean of zero and unknown standard deviation (*sigma_alpha*) for the discrimination parameter. Consequently, the parameters block includes three lines for these three hyperparameters (Listing 1, lines 11-13) as follows:

```
6   parameters {
7   vector[n_student] theta;
8   vector <lower=0 > [n_item] alpha;
9   vector[n_item] beta;
10  vector<lower=0,upper=1 > [n_item] gamma;   //item pseudo-guessing
11  real mu_beta;
12  real<lower=0 > sigma_alpha;
13  real<lower=0 > sigma_beta;
14  }
```

The graded response model (GRM; Samejima, 1969) can be viewed as a generalized case of the 2PL IRT model in that it allows more than two response categories in an item. Instead of directly modeling the probability of a response of a certain response category, GRM first models the probability of responding below a certain category versus above that category. The difference of such probabilities is the probability of responding at that category. The mathematical equation for GRM is given as

$$p_{ij}(u_{ij}=k|\theta_i,a_j,b_{jk})=\frac{1}{1+\exp(-a_j(\theta_i-b_{jk}))}-\frac{1}{1+\exp(-a_j(\theta_i-b_{jk+1}))}, \quad (2)$$

where k refers to the kth category, b_{jk} is the category difficulty of item j, and the other terms remain the same as in Equation (1). The probability of responding below the first category is set to be 0, and the probability of responding above the highest category is 1.

Similarly, all the GRM model parameters on the right side of Equation (2) are specified in this block (Listing 2, lines 8-10). The prior distribution for item difficulty contains two hyperparameters, and they are specified in this block as well (Listing 2, lines 11-12). The category difficulty parameter κ is declared as *ordered* (Listing 2, line 10), a special data type which conveniently constrain κ to be a vector whose elements are arranged in order of increasing value. This is similar to the function *ranked* used in BUGS for the GRM estimation (e.g., Curtis, 2010) but more efficient.

The nominal response model (NRM; Bock, 1972) takes the form

$$p_{ij}(u_{ij}=k|\theta_i,a_{jk},\gamma_{jk})=\frac{\exp(a_{jk}\theta_i+\gamma_{jk})}{\sum_{h=1}^{c}\exp(a_{jh}\theta_i+\gamma_{jh})}, \quad (3)$$

where a_{jk} and γ_{jk} are the slope and intercept parameters of item j on category k, and the other terms remain the same. As no hyperparameters are set in the priors for any of the parameters on the right side of Equation (3), only three lines of codes are needed in the parameters block for the NRM (Listing 3, lines 8-10).

The data type for all parameters in the parameters block need to be specified. If one of the model parameters is defined as a vector or a matrix, its dimension needs to be specified. For example, in the 3PL IRT model, the ability parameter (*theta*) is specified to be a vector with a length equal to the number of students (Listing 1, line 7). In addition, if a parameter is expected to be within a certain range, the range should be provided right after the data type. For example, the pseudoguessing parameter of the 3PL model (*gamma*) is constrained to range from 0 to 1 (Listing 1, line 10). In the GRM, the standard deviation of the prior for the category difficulty parameter (*sigma_kappa*) is constrained to be nonnegative (Listing 2, line 12).

Model Block

The model block, where the priors and the model are specified, is the most essential component of a Stan program. Note that all the parameters appearing in this block have to be specified in advance in the parameters block, or Stan will produce an error message and stop. Prior distributions for all the parameters in the parameter block need to be specified. If a prior is not specified, a uniform prior will be automatically assumed by Stan. For example, we use a standard normal distribution as a prior for the ability parameters (*theta*) in the 3PL model (Listing 1, line 16), and a normal distribution with unknown mean (*mu_beta*) and unknown standard deviation (*sigma_beta*) for the item difficulty parameters (*beta*; Listing 1, line 17). The hyperprior for *mu_beta* is specified as a normal distribution with a mean of 0 and standard deviation of 5 (Listing 1, line 18), and a Cauchy distribution as a hyperprior for *sigma_beta* (Listing 1, line 19). Note that since *sigma_beta* has been constrained to be nonnegative in the parameters block (Listing 1, line 13), this Cauchy distribution as a hyperprior is actually a half Cauchy distribution. Similar to any other Bayesian software program, the priors may be informative, weakly informative, or noninformative, and the choice should reflect our belief about those parameters.

```
15  model {
16  theta ~ normal(0,1);
17  beta ~ normal(mu_beta,sigma_beta);
18  mu_beta ~ normal(0,5);
19  sigma_beta ~ cauchy(0,5);
20  alpha ~ lognormal(0,sigma_alpha);
21  sigma_alpha ~ cauchy(0,5);
22  gamma ~ beta(5,23);
23  for(i in 1:n_student){
24  for (j in 1:n_item){
25  real p;//create a local variable within the loop to make Stan code more readable
26  p= inv_logit(alpha[j]*(theta[i] - beta[j]));
27  Y[i,j] ~ bernoulli(gamma[j] + (1-gamma[j])*p);
28  }}
29  }
```

After all priors and hyperpriors have been specified, the model is specified in the sampling statement through a function. Three functions, including *bernoulli* (Listing 1, line 27), *ordered_logistic* (Listing 2, line 25), and *categorical_logit* (Listing 3, line 29), are used for the 3PL model, the GRM, and the NRM, respectively. Since the *bernoulli* function is the Stan analog of the *dbern* function in BUGS, we focus our discussion on *ordered_logistic* and *categorical_logit* here.

The function *categorical_logit* is a direct parameterization of *categorical*, which is the Stan counterpart of the function *dcat* in BUGS. Such a parameterization is ''numerically more stable if the chance of success parameter is on the logit scale . . .'' (Stan Development Team, 2016a, p. 409). The function *categorical_logit* can be used for all polytomous IRT models that belong to the divide-by-total family. For the NRM, *categorical_logit(zetan[j] + lambdan[j]*theta[i])* (Listing 3, line 29) is equivalent to *categorical(**p**)*, where **p** is a simplex of length J (a vector of nonnegative elements that sum to 1) and the jth element of **p** can be expressed as

$$p_j = \frac{\exp(\text{zetan[j]} + \text{lambdan[j]} * \text{theta[i]})}{\sum_{h=1}^{J} \exp(\text{zetan[h]} + \text{lambdan[h]} * \text{theta[i]})}, \tag{4}$$

where the terms other than notational differences, are the same as those in Equation (3).

The function *ordered_logistic* in the GRM (Listing 2, line 25), combined with the category difficulty parameter *kappa* specified as the *ordered* data type in the parameters block (Listing 2, line 10), leads to a mathematical equation that is identical to Equation (2), with the category difficulty parameter b_{jk} replaced by *kappa[j]*.

Generated Quantities Block

The generated quantities block is an optional component of the Stan code, and it is usually used when there is a need to compute new variables and obtain their corresponding posterior distributions. In the IRT context, this block can be used to compute the model-based log-likelihood, which is used for the computation of model fit indices for model comparison and selection purposes. The computation of model-based log-likelihood requires the use of a function that is the combination of the function used in the sampling statement and a suffix ''*log*.'' For example, in the generated block section of the 3PL model (Listing 1, lines 30-38), the function *bernoulli_log* (Listing 1, line 36) is used, which is the function *bernoulli* used in the sampling statement of the model block (Listing 1, line 27) plus the suffix ''*log*.'' The generated quantity block included for all the three IRT models in this article can be deleted if model comparison and selection is not of interest.

With the log-likelihood specified in this block, their posterior distributions can be obtained and used to compute the widely available information criterion (WAIC; Watanabe, 2010) and leave-one-out cross-validation (LOO; Vehtari, Gelman, & Gabry, 2016a). In BUGS, model comparison is usually conducted using the deviance

```
generated quantities {
vector[n_item] log_lik[n_student];
for (i in 1: n_student){
for (j in 1: n_item){
real p;
p= inv_logit(alpha[j]*(theta[i] -beta[j]));
log_lik[i, j] = bernoulli_log(Y[i, j], gamma[j] + (1-gamma[j])*p);
}}
}
```

information criterion (DIC; Spiegelhalter, Best, Carlin, & van der Linde, 2002). Stan does not compute DIC for model comparison and selection purposes; instead, it uses WAIC and LOO, which are fully Bayesian and theoretically superior to traditional information-based model selection criteria such as the Akaike information criterion (AIC; Akaike, 1974), the Bayesian information criterion (BIC; Schwarz, 1978), and DIC. In the context of IRT model selection, Luo and Al-Harbi (2016) investigated the performances of WAIC and LOO in choosing the correct dichotomous IRT models and found that they were superior to the more traditional methods such as the likelihood ratio test, AIC, BIC, and DIC. The computation of WAIC and LOO can be intensive and nontrivial, and approximation methods usually have to be used. Fortunately, an R package **loo** (Vehtari, Gelman, & Gabry, 2016b) has been developed and can be used in combination with the **rstan** package to compute WAIC and LOO. In the ''Illustrations for Unidimensional Dichotomous, Multidimensional Testlet and Multilevel IRT Models'' section, we will demonstrate how to compare and choose competing IRT models with the R packages **rstan** and **loo**.

Transformed Parameters Block

The transformed parameters block is optional too and used when some parameters specified in the parameters block need to be transformed. A transformed parameter in this block is usually constrained to be a function of some other parameters in the parameters block, and imposing a prior distribution on such a constrained parameter in the subsequent model block causes Stan to produce an error message and stop, which is another notable difference between Bugs and Stan. Consequently, it is necessary in Stan to separate the freely estimated parameters and the constrained parameters into the parameters block and the transformed parameters block, respectively.

In the IRT context, this block is often used for model identification purposes. For example, models that are members of the Rasch family can use this block to constrain the sum of item difficulty parameters to be 0. Among the three IRT models illustrated in this article, the NRM is the only one that requires the transformed parameter block to identify the model by constraining both the sum of category intercepts and that of category slopes for each item to be 0. One way to set the sum of all

intercept parameters for an item to be 0 is to constrain the last intercept parameter to be the negative sum of the remaining intercept parameters of that particular item. In the example code, a different approach (e.g., Bolt, Cohen, & Wollack, 2001) is followed by creating new sets of intercept and slope parameters which are deviations from the means of the original intercept and slope parameter specified in the parameters block (Listing 3, lines 17-18) as the following presented lines. Note that while the transformed parameters (*lambdan* and *zetan*) are directly used in the sampling statement to describe the model of interest (Listing 3, line 29), only the parameters in the parameters block from which they are transformed (*lambda* and *zeta*) are given priors (Listing 3, lines 23-26).

```
12    transformed parameters {
13    vector[K] zetan[n_item];   //intercept with constraints
14    vector[K] lambdan[n_item];   //slope with constraints
15    for (k in 1:n_item) {
16    for (l in 1:K) {
17    zetan[k,l]=zeta[k,l]-mean(zeta[k]);   //constrain intercept sum for each item to 0
18    lambdan[k,l]=lambda[k,l]-mean(lambda[k]);   //constrain slope sum for each item to 0
19    }}
20    }
```

Illustrations for Unidimensional Dichotomous, Multidimensional Testlet and Multilevel IRT Models

In this section, both real and simulated data sets are used to demonstrate how to use Stan to analyze dichotomous item response data using unidimensional IRT model, multidimensional testlet model, and multilevel IRT model. We assume that the **rstan** package has been installed (the installation of **rstan** is the same as any other R packages on the R website https://www.r-project.org). The first example involves model selection among three competing dichotomous IRT models. The second example is a demonstration of using a multidimensional extension of the Rasch model to estimate testlet effects with a simulated data set. In the last example, Stan is used to fit a multilevel 3PL IRT model to a learning outcome assessment data set. The second and the third examples are intended to demonstrate the flexibility of Stan to extended multidimensional and multilevel IRT models.

Example 1: Model Comparison

The data set used in this example is from a high-stakes English proficiency test in a Middle-Eastern country for admission and placement purposes, consisting of four sections including listening, reading comprehension, sentence completion, and grammar. All items are multiple-choice questions. For illustration of model comparison,

we only used response data from the listening section which contains 20 items with a sample size of 1,637.

The code for conducting model comparison is provided in Listing 4. First the working directory in R is set up and the **rstan** package is loaded. Then it is specified that a compiled Stan program is saved to the hard disk so that it will not be compiled again, and Stan is requested to run in parallel (Listing 4, lines 1-4). Note that line 4 in Listing 4 only applies if **rstan** runs on a local multicore computer: It allows multiple chains to be simultaneously processed by multiple processors, given that the number of processors are equal to or greater than that of chains. The R package **loo** (Listing 4, line 5) is also loaded for the computation of WAIC and LOO.

Assuming the data set has been already saved as a comma-separated values (csv) file in the directory folder, the data are imported and its dimension (which indicates the number of students and items in the data set) is extracted (Listing 4, lines 6-8). A data list is created to include the number of students, the number of items, and the response matrix (Listing 4, line 9). It should be noted that the name assigned in the data list must match what is provided in the data block of a Stan program. Otherwise Stan will produce an error message. For example, here the response matrix is named as *Y*, and in the Stan code for the 3PL IRT model the same *Y* is found (Listing 1, line 4).

Four arguments for the function *stan* in **rstan** are necessary to run the program. The first argument is *file*, which requires a user to specify a Stan program file saved in advance. In this example, three Stan program files, *irt_1pl.stan, irt_2pl.stan, and irt_3pl.stan* for the 1PL, 2PL, and 3PL IRT models, respectively, are assumed to exist in the directory folder, so the *file* argument in the function *stan* (Listing 4, lines 10-12) can locate the corresponding Stan program file. The second necessary argument is *data*, referring to the data list created earlier. The third and four arguments specify the number of chains and the number of iterations within each chain. For dichotomous IRT models, Stan needs approximately 200 iterations to reach model convergence (convergence check is elaborated below). In this example, three chains with 1,000 iterations per chain were run. Note that since the number of burn-in iterations is not specified through the argument *warmup*, as default the first 500 iterations were automatically discarded as burn-in iterations.

Running **rstan** with the function *stan* produces an object of S4 class. This object contains information regarding the posterior distributions of model parameters. For example, the object of fitting a 3PL IRT model can be saved as *irt_3pl* (Listing 4, line 12) to check model convergence. Directly printing *irt_3pl* in the R console provides summary information regarding posterior distributions of all parameters. For illustration, only the item discrimination parameters in the 3PL IRT model are printed in line 13 of Listing 4. Figure 1 is the screenshot of the R console after running this line of code. Note that this output is very similar to what is provided by R2WinBUGS (Sturtz, Ligges, & Gelman, 2005). A list of summary statistics is provided for the posterior distribution of every parameter. The posterior mean and the standard deviation can be used as the point estimates of the discrimination parameters and their standard errors. The column named *n_eff* lists the effective number of simulation draws, which

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
alpha[1]	1.42	0.00	0.13	1.17	1.33	1.42	1.51	1.70	982	1.00
alpha[2]	1.52	0.00	0.13	1.29	1.44	1.52	1.60	1.79	927	1.00
alpha[3]	2.78	0.02	0.40	2.11	2.50	2.73	3.01	3.66	654	1.00
alpha[4]	1.07	0.00	0.13	0.85	0.97	1.06	1.15	1.35	1160	1.00
alpha[5]	0.62	0.00	0.11	0.44	0.54	0.60	0.68	0.87	1043	1.00
alpha[6]	1.72	0.01	0.21	1.35	1.58	1.71	1.85	2.18	738	1.01
alpha[7]	2.31	0.02	0.40	1.63	2.04	2.28	2.55	3.17	705	1.00
alpha[8]	0.94	0.01	0.16	0.68	0.82	0.92	1.02	1.29	950	1.00
alpha[9]	2.10	0.01	0.42	1.38	1.82	2.07	2.35	3.03	1065	1.00
alpha[10]	2.27	0.01	0.42	1.55	1.98	2.23	2.51	3.15	849	1.00
alpha[11]	2.31	0.02	0.54	1.44	1.92	2.23	2.62	3.56	1083	1.00
alpha[12]	1.11	0.00	0.12	0.90	1.03	1.10	1.18	1.38	1500	1.00
alpha[13]	2.31	0.02	0.40	1.65	2.01	2.26	2.55	3.22	650	1.00
alpha[14]	2.65	0.02	0.56	1.76	2.27	2.58	2.95	3.98	945	1.00
alpha[15]	1.62	0.01	0.35	1.00	1.37	1.60	1.84	2.36	927	1.00
alpha[16]	2.80	0.02	0.67	1.75	2.34	2.71	3.16	4.29	730	1.00
alpha[17]	1.31	0.01	0.24	0.91	1.15	1.29	1.45	1.82	867	1.00
alpha[18]	1.75	0.02	0.50	0.86	1.42	1.71	2.05	2.85	540	1.00
alpha[19]	3.52	0.03	0.77	2.29	2.97	3.43	3.94	5.36	820	1.01
alpha[20]	3.08	0.02	0.52	2.19	2.71	3.05	3.40	4.23	551	1.00

Figure 1. Estimates of item discrimination parameters in the three-parameter logistic item response theory (3PI IRT) model.

can be viewed as the effective sample for a posterior distribution on which inferences are based. The last column *Rhat* lists the Gelman and Rubin's convergence diagnostic (Gelman & Rubin, 1992), a popular statistic that computes the potential scale reduction factor (PSRF). A PSRF value close to 1 usually indicates model convergence, and the PSRF value of 1.1 has been recommended as a threshold with smaller values indicating model convergence (Gelman et al., 2014, p. 287). As can be seen in Figure 1, the largest PSRF value for the item discrimination parameter is 1.01. As all the other parameters not printed here have PSRF values smaller than 1.05, it was concluded that model convergence had been reached.

The trace plots for model parameters can be requested with the function *traceplot* in **rstan** for additional convergence check (e.g., Listing 4, line 14). For illustrative purposes, Figure 2 lists the trace plots for all 20 pseudoguessing parameters in the 3PL IRT model. It should be noted that those plots only show the last 500 iterations as *inc_warmup = FALSE* and therefore the first 500 iterations are excluded. Usually the appearance of a ''fat hairy caterpillar'' (Lunn, Jackson, Best, Thomas, & Spiegelhalter, 2012, p. 73) in trace plots as shown in Figure 2 indicates convergence.

In general, **rstan** allows easy use of many other model convergence diagnostics through the R package **ShinyStan** (Stan Development Team, 2016b). Interested readers are referred to Cowles and Carlin (1996) for a review of MCMC convergence diagnostics and Sinharay (2004) for their application in the IRT context.

After confirming model convergence, WAIC and LOO are computed in lines 15 to 23 in Listing 4 for each compared model. As the log-likelihood has been computed in the generated quantity block, the function *extract_log_lik* can be used to directly extract the posterior distribution of the log-likelihood (Listing 4, lines 15, 18, and 21) and then WAIC and LOO are computed with the functions *waic* and *loo*, respectively (e.g., Listing 4, lines 16-17). Table 1 lists the WAIC and LOO values for the three

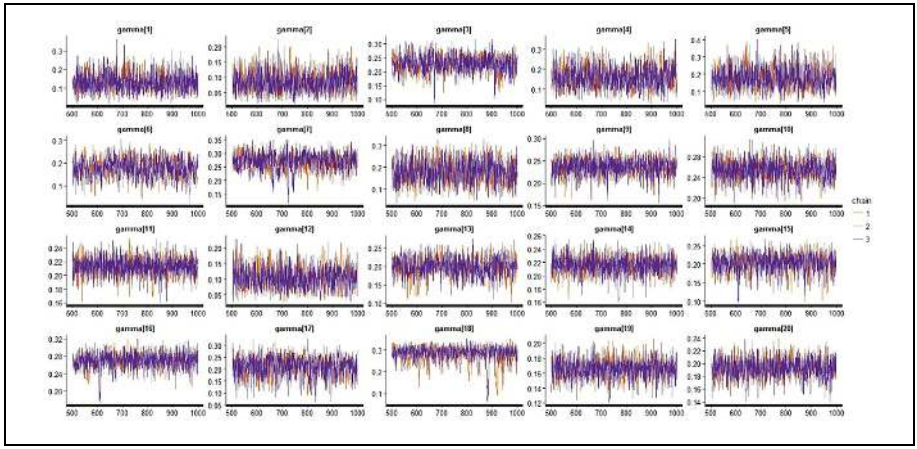

Figure 2. Trace plots of the pseudoguessing parameters in the three-parameter logistic item response theory (3PL IRT) model.

Table 1. WAIC and LOO for Three Dichotomous Item Response Theory Models.

	1PL	2PL	3PL
WAIC	44,667	44,252	43,707
LOO	44,666	44,263	43,729

Note. WAIC = widely available information criterion; LOO = leave-one-out cross-validation; 1PL = one-parameter logistic; 2PL = two-parameter logistic; 3PL = three-parameter logistic.

dichotomous IRT models, both of which lead to the same conclusion that the 3PL IRT model has the smallest values of LOO and WAIC. Consequently, it was concluded that the 3PL IRT model was the best-fitting model among the three competing dichotomous IRT models.

Example 2: The Rasch Testlet Model

This example demonstrates how to use **rstan** to fit the Rasch testlet model (Wang & Wilson, 2005), a multidimensional extension of the Rasch model. The Rasch testlet model for data generation is given as follows:

$$p_i(\theta_j) = \frac{1}{1 + \exp\left(-(\theta_j - b_i + \gamma_{jd(i)})\right)}, \tag{5}$$

where $\gamma_{jd(i)}$ is a person-specific testlet effect parameter that models the interaction between an examinee and a testlet, and all the other terms remain the same as in

Equation (1). The magnitude of the testlet variance ($\sigma^2_{\gamma_{jd(i)}}$) indicates the magnitude of the testlet effect. To provide data to fit the Rasch testlet model, a data set of 2,000 examinees' responses to a test of 36 items was simulated. These 36 items formed six testlets, and the testlet variance $\sigma^2_{\gamma_{jd(i)}}$ was assumed to be 1 for all six testlets, representing a large testlet effect.

Listing 5 provides the R code for the analysis in this example. The data are transposed and the number of items and the number of examinees are extracted in lines 6 to 8 in Listing 5. Such data transposing makes Stan more efficient as illustrated below. The number of testlets or item groups is specified in a data list named *data_testlet* in lines 9 to 11 of Listing 5.

Stan code for the Rasch testlet model can be saved as an external file and runs in the *stan* function. Another way is to specify the Stan program for the Rasch testlet model as an object called *code_testlet* in the R environment as in lines 12 to 45 of Listing 5. This Stan program is similar to that for the 3PL IRT model except the inclusion of the number of testlets and the item group membership as specified in lines 15 to 16 of Listing 5 in the data block. In the parameter block, two parameters, *gamma* and *sd_gamma* are specified for the testlet parameters and the testlet variance, respectively (Listing 5, lines 21 and 24). In the model block, normal distributions are assumed as priors for $\gamma_{jd(i)}$ with the means fixed to be 0, and a half Cachy distribution as the prior for each $\sigma_{\gamma_{jd(i)}}$ (Listing 5, lines 37-40).

This example uses the transformed parameters block to constrain the sum of item difficulty parameters to be zero (Listing 5, line 30) for model identification. In addition, the function *bernoulli_logit* is used (Listing 5, line 42) in the sampling statement. This function is a direct parameterization of *bernoulli* and can be used for both 1PL and 2PL IRT models. For the 2PL IRT model, it is expressed as *bernoulli_logit(alpha[j]*(theta[i]- beta[j]))*, which is equivalent to *bernoulli(1/(1 + exp(-alpha[j]*(theta[i]- beta[j]))))*. For the 3PL IRT model, the function *bernoulli_logit* cannot be used due to the addition of the lower-asymptote parameter.

Different from that specified for the above three IRT models, the sampling statement for the Rasch testlet model (Listing 5, lines 41-43) is vectorized for efficiency. In the sampling statement for the first three models (e.g., Listing 1, lines 22-25), a double loop is used to model one element at a time from the response matrix. This example demonstrates that a single loop is used to model one vector (its length is the number of examinees) at a time from the response matrix. Such a vectorized sampling statement improves computation efficiency in Stan because ''. . . vectorized log probability functions are faster than their equivalent form defined with loops'' (Stan Development Team, 2016a, p. 405).

The sampling statement in this example (Listing 5, line 42) warrants some additional explanation. With the data transposed (Listing 5, line 6), *res[i]* outputs the vector of all examinees' responses to item *i* with a length equal to the number of examinees. The parameter *theta* is defined as a vector with the same length (Listing 5, line 20), and *gamma[t_index[i]]* indicates a specific testlet that item *i* is associated with and the vector of the testlet effect with the same length as the number of

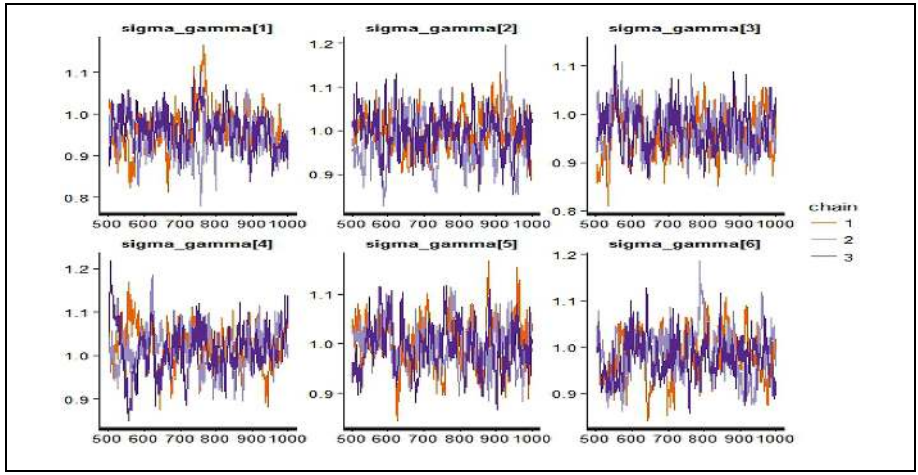

Figure 3. Trace plots of testlet effect parameters in the Rasch testlet model.

examinees. Although *beta[i]* is a scalar, it is automatically transformed by Stan into a vector of the same length to be compatible with *theta* and *gamma[t_index[i]]*.

With the Stan program for the Rasch testlet model defined as an R object called *code_testlet*, the Rasch testlet model parameters can be estimated using the function *stan* (Listing 5, line 46). It should be noted that when the Stan code is directly written in the R environment, *model_code* instead of *file* should be used as the first argument in the function *stan* to refer to *code_testlet*.

Evidently different from the Gibbs sampler and the Metropolis algorithm that require a larger number of iterations for the Rasch testlet model to converge, the efficient HMC algorithm implemented in Stan needs fewer than 500 iterations to converge. In this example, 1,000 iterations were run and the first 500 iterations were discarded as burn-in iterations. The whole estimation process only took approximately 15 minutes on a desktop computer with an Intel Xeon E5 processor and as will be shown next, model convergence was achieved within the first 500 iterations.

Figure 3 shows the trace plots of the testlet effect parameters. Good mixing is observed for all six testlet effect parameters, indicating that convergence has been reached for these parameters. Figure 4 shows the screenshot of the testlet effect parameter estimates printed in the R console. The PSRF values for the six testlet effect parameters are all below 1.1, with the largest value of 1.05. Though not listed here, all the other model parameters have PSRF values no greater than 1.05, with good mixing as shown in the trace plots. The estimated testlet variance parameters are all close to the generating value of 1, suggesting that they have been accurately estimated by Stan. Consequently, it was concluded that convergence was achieved for the Rasch testlet model.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
sigma_gamma[1]	0.95	0.01	0.05	0.86	0.92	0.95	0.98	1.05	87	1.03
sigma_gamma[2]	0.99	0.00	0.05	0.90	0.96	0.99	1.03	1.09	149	1.02
sigma_gamma[3]	0.98	0.00	0.04	0.89	0.95	0.98	1.01	1.06	131	1.03
sigma_gamma[4]	1.01	0.00	0.04	0.93	0.98	1.01	1.04	1.10	157	1.03
sigma_gamma[5]	1.00	0.00	0.05	0.91	0.97	1.00	1.03	1.09	105	1.03
sigma_gamma[6]	0.99	0.01	0.05	0.90	0.95	0.99	1.02	1.09	88	1.05

Figure 4. Testlet effect parameter estimates in the Rasch testlet model.

Example 3: A Multilevel 3PL IRT Model

This example demonstrates how to fit a multilevel 3PL IRT model in Stan. The data set is an end-of-program learning outcome assessment test for students enrolled in engineering programs in a Middle-Eastern country. The purpose of the analysis is on evaluating those programs based on program mean scores. A traditional approach to computing program mean scores is to calibrate items and score students with an IRT model and average individual scores within a program to obtain the program mean score. However, such an approach ignores the measurement error inherent in individual scores. An alternative, as proposed by Fox and Glas (2001), is to use a multilevel IRT model that simultaneously estimates item parameters, individual scores, and program mean scores. This example fits a 3PL multilevel IRT model that takes into consideration the measurement error in individual scores to estimate program mean scores. The multilevel 3PL IRT model is expressed as follows:

$$p_i(\theta_{gj}) = c_i + \frac{1 - c_i}{1 + \exp(-a_i(\theta_{gj} + \theta_g - b_i))}, \tag{6}$$

where θ_{gj} is the individual score of student j in group g, θ_g is program g's mean score, and all the other terms remain the same as in Equation (1).

The R code for this example is provided in Listing 6. The data set consists of 43 columns, with the first 42 for item responses and the last one for the group membership for each person. Item responses and the group membership are imported in lines 6 and 9 of Listing 6, and the number of items, examinees, and programs are extracted in lines 7, 8, and 10 of Listing 6. The Stan program for the multilevel 3PL IRT model is defined as an R object named *code_m3pl* (Listing 6, lines 12-46). To avoid repetition, the illustration of this example focuses on additional Stan codes in Listing 6 related to the multilevel structure.

The group membership information and the number of groups are included in the data block (Listing 6, lines 15-16). In the parameter block, two lines are added for the program mean score parameter (*s_theta*) and its standard deviation (*sigma_school*; Listing 6, lines 21 & 28). In the model block, a normal distribution with a mean of 0 and unknown standard deviation (*sigma_school*) is specified as the prior for *s_theta* (Listing 6, line 35) whereas a half Cauchy distribution is set as the hyper priors for the parameter *sigma_school* (Listing 6, line 39). The sampling statement (Listing 6,

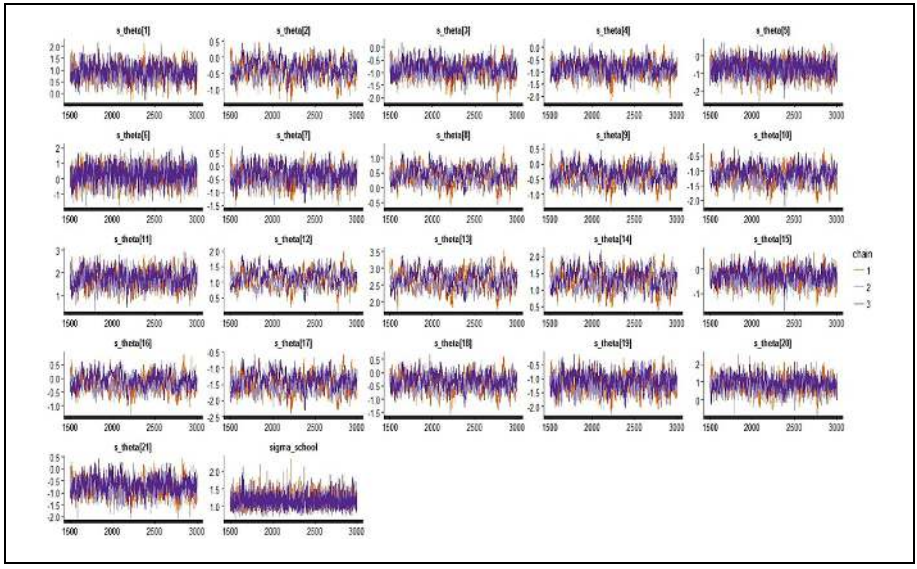

Figure 5. Trace plots of program effect parameters in the multilevel three-parameter logistic item response theory (3PL IRT) model.

lines 40-44) in this example is similar to that of the 3PL IRT model (Listing 1, lines 33-37), with the addition of the program mean score parameter *s_theta[g[i]]*. In this example, a double loop is still used instead of a single loop as the function *bernoulli* does not allow vectorization for the 3PL model.

For this model, 1,000 iterations are sufficient for convergence. To be conservative, three chains were run with 3,000 iterations for each chain (Listing 6, line 47) and as default, the first 1,500 iterations were discarded as burnin and subsequent inferences were based on the last 1,500 iterations in each chain. The computation took 45 minutes on a desktop computer with an Intel Xeon E5 processor. Based on our experience, BUGS may need tens of thousands of iterations and hours of running time to estimate parameters for this model. Such a noticeable difference in computation time confirms the efficiency of the HMC algorithm implemented in Stan over the traditional MCMC algorithms.

The trace plots for *s_theta* and *sigma_school* are requested, as well as the summary statistics of the posterior distributions (Listing 6, lines 48-49). Figures 5 and 6 show the trace plots and the screenshot of the parameter estimates printed in the R console. As can be seen, good mixing is observed for all trace plots in Figure 5, and the PSRF values for all parameters shown in Figure 6 are all below 1.1, with the largest value being 1.02. Though not listed here, all the other model parameters have similar trace plots and PSRF values no greater than 1.05, indicating that convergence has been reached. The 95% credible interval for each *s_theta* allows to determine whether

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
s_theta[1]	0.84	0.02	0.36	0.12	0.60	0.85	1.09	1.55	229	1.01
s_theta[2]	-0.46	0.02	0.29	-1.03	-0.64	-0.46	-0.27	0.08	140	1.01
s_theta[3]	-0.95	0.03	0.33	-1.61	-1.16	-0.93	-0.73	-0.35	166	1.01
s_theta[4]	-0.95	0.02	0.32	-1.59	-1.16	-0.94	-0.75	-0.36	169	1.01
s_theta[5]	-0.77	0.02	0.46	-1.69	-1.07	-0.76	-0.45	0.10	450	1.00
s_theta[6]	0.14	0.02	0.58	-1.01	-0.25	0.15	0.53	1.28	597	1.00
s_theta[7]	-0.44	0.02	0.34	-1.12	-0.67	-0.44	-0.20	0.22	226	1.01
s_theta[8]	0.33	0.02	0.29	-0.27	0.15	0.35	0.53	0.86	142	1.01
s_theta[9]	-0.45	0.02	0.29	-1.03	-0.63	-0.44	-0.25	0.10	149	1.01
s_theta[10]	-1.20	0.02	0.30	-1.79	-1.39	-1.19	-1.00	-0.64	155	1.01
s_theta[11]	1.67	0.02	0.38	0.90	1.43	1.68	1.92	2.39	242	1.01
s_theta[12]	1.06	0.02	0.28	0.52	0.87	1.07	1.24	1.56	126	1.02
s_theta[13]	2.57	0.03	0.28	2.03	2.39	2.58	2.76	3.12	126	1.02
s_theta[14]	1.22	0.03	0.30	0.63	1.03	1.24	1.42	1.78	138	1.01
s_theta[15]	-0.39	0.02	0.34	-1.08	-0.61	-0.38	-0.17	0.27	202	1.01
s_theta[16]	-0.21	0.02	0.30	-0.81	-0.41	-0.20	0.00	0.36	154	1.01
s_theta[17]	-1.53	0.02	0.29	-2.13	-1.71	-1.52	-1.33	-0.99	148	1.01
s_theta[18]	-0.45	0.02	0.32	-1.09	-0.65	-0.44	-0.23	0.15	164	1.02
s_theta[19]	-1.22	0.02	0.33	-1.89	-1.42	-1.21	-0.99	-0.59	183	1.01
s_theta[20]	0.82	0.02	0.41	-0.01	0.55	0.83	1.10	1.59	325	1.01
s_theta[21]	-0.83	0.02	0.37	-1.57	-1.06	-0.81	-0.58	-0.12	227	1.01
sigma_school	1.17	0.01	0.21	0.84	1.02	1.14	1.29	1.67	1394	1.00

Figure 6. Program effect parameter estimates in the multilevel three-parameter logistic item response theory (3PL IRT) model.

a particular program has a mean score significantly different from zero (the grand mean score). Take Program 11 as an example: Since *s_theta[11]* has a posterior mean of 1.67 with a 95% credible interval of (0.90, 2.39), this program has a program effect significantly above the average and is therefore classified as effective. The estimated standard deviation of the program effect is 1.17 with a 95% credible interval of (0.84, 1.67), suggesting that in this data set, there is substantial between-program variation.

Summary

Stan is a relatively new programming software that implements the powerful HMC algorithm. To date, there is no systematic introduction to using Stan for Bayesian IRT model parameter estimation. This article introduces Stan programs for estimating common dichotomous and polytomous IRT models. Furthermore, it demonstrates IRT model comparison with two R packages **rstan** and **loo**, using methods that are fully Bayesian and hence theoretically superior to DIC. Estimation of multidimensional and multilevel extensions of dichotomous IRT models using the Stan program is also demonstrated. In general, Stan is efficient and requires considerably shorter time than other Bayesian software programs for estimating these complex models.

Stan is not without limitations. While WinBUGS can automatically generate values from the posterior predictive distribution for missing response data, Stan treats all missing data as parameters and consequently, a Stan user has to define them in the parameters block or transformed parameters block. With the combination of nonsystematic missing patterns and complex models, such procedures can become tedious and error-prone. Stan also lacks support for sampling discrete parameters,

and such parameters need to be marginalized out in order to be estimated in Stan. Although there are some sources such as the Stan User Manual that provide some introduction to how to conduct the marginalization, this procedure may require a substantial amount of coding effort and can be difficult to implement to some applied researchers and practitioners. In the psychometric context, such a disadvantage translates into the fact that mixture IRT models and latent class models cannot be directly specified in Stan, since both contain discrete latent variables. In contrast, BUGS allows for straightforward specifications of such models.

It is hoped that this tutorial will introduce Stan to interested researchers as a powerful and efficient software program suitable for Bayesian IRT analysis. The introduction to the common dichotomous and polytomous IRT models and their multidimensional and multilevel extensions will reduce the learning curve of Stan and facilitate model parameter estimation for further extended IRT models. For researchers who want to fit more complex IRT models to their data using MCMC algorithm but are not satisfied with the computational speed of the Gibbs sampler and Metropolis algorithm in BUGS and other MCMC packages, Stan may be an attractive alternative.

Appendix

Listing 1: Stan Code for the 3PL IRT Model

```
1   data {
2   int<lower=0> n_student;
3   int<lower=0> n_item;
4   int<lower=0,upper=1> Y[n_student,n_item];
5   }
6   parameters {
7   vector[n_student] theta;
8   vector <lower=0> [n_item] alpha;
9   vector[n_item] beta;
10  vector<lower=0,upper=1> [n_item] gamma;   //item pseudo-guessing
11  real mu_beta;
12  real<lower=0> sigma_alpha;
13  real<lower=0> sigma_beta;
14  }
15  model {
16  theta ~ normal(0,1);
17  beta ~ normal(mu_beta,sigma_beta);
18  mu_beta ~ normal(0,5);
19  sigma_beta ~ cauchy(0,5);
20  alpha ~ lognormal(0,sigma_alpha);
21  sigma_alpha ~ cauchy(0,5);
22  gamma ~ beta(5,23);
23  for(i in 1:n_student){
24  for (j in 1:n_item){
```

(continued)

Listing 1: (continued)

```
real p;    //create a local variable within the loop to make Stan code more readable
p= inv_logit(alpha[j]*(theta[i] - beta[j]));
Y[i,j] ~ bernoulli(gamma[j] + (1-gamma[j])*p);
}}
}
generated quantities {
vector[n_item] log_lik[n_student];
for (i in 1: n_student){
for (j in 1: n_item){
real p;
p= inv_logit(alpha[j]*(theta[i] -beta[j]));
log_lik[i, j] = bernoulli_log(Y[i, j], gamma[j] + (1-gamma[j])*p);
}}
}
```

Listing 2: Stan Code for the Graded Response Model

```
data{
int<lower=2, upper=4 > K;    //number of categories
int <lower=0 > n_student;
int <lower=0 > n_item;
int<lower=1,upper=K > Y[n_student,n_item];
}
parameters {
vector[n_student] theta;
real<lower=0 > alpha [n_item];
ordered[K-1] kappa[n_item];    //category difficulty
real mu_kappa;    //mean of the prior distribution of category difficulty
real<lower=0 > sigma_kappa;    //sd of the prior distribution of category difficulty
}
model{
alpha ~ cauchy(0,5);
theta ~ normal(0,1);
for (i in 1: n_item){
for (k in 1:(K-1)){
kappa[i,k] ~ normal(mu_kappa,sigma_kappa);
}}
mu_kappa ~ normal(0,5);
sigma_kappa ~ cauchy(0,5);
for (i in 1:n_student){
for (j in 1:n_item){
Y[i,j] ~ ordered_logistic(theta[i]*alpha[j],kappa[j]);
}}
}
generated quantities {
vector[n_item] log_lik[n_student];
for (i in 1: n_student){
for (j in 1: n_item){
log_lik[i, j] = ordered_logistic_log (Y[i, j],theta[i]*alpha[j],kappa[j]);
}}
}
```

Listing 3: Stan Code for the Nominal Response Model

```
data{
int<lower=2, upper=4 > K;
int <lower=0 > n_student;
int <lower=0 > n_item;
int<lower=1,upper=K > Y[n_student,n_item];
}
parameters {
vector[K] zeta[n_item];    //freely estimated intercept
vector[K] lambda[n_item];    //freely estimated slope
vector[n_student] theta;
}
transformed parameters {
vector[K] zetan[n_item]; //intercept with constraints
vector[K] lambdan[n_item]; //slope with constraints
for (k in 1:n_item) {
for (l in 1:K) {
zetan[k,l]=zeta[k,l]-mean(zeta[k]); //constrain intercept sum for each item to 0
lambdan[k,l]=lambda[k,l]-mean(lambda[k]); //constrain slope sum for each item to 0
}}
}
model{
theta ~ normal(0,1);
for (i in 1: n_item){
zeta[i] ~ normal(0,2);
lambda[i] ~ normal(0,2);
}
for (i in 1:n_student)
for (j in 1:n_item)
Y[i,j] ~ categorical_logit(zetan[j] + lambdan[j]*theta[i]);
}
generated quantities {
vector[n_item] log_lik[n_student];
for (i in 1: n_student){
for (j in 1: n_item){
log_lik[i, j] = categorical_logit_log (Y[i, j], zetan[j] + lambdan[j]*theta[i]);
}}
}
```

Listing 4: R Code for Example 1, Model Selection

```
setwd("C:\\ Rstan Tutorial")
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
library(loo)
data<-read.table("example1.csv",header=FALSE,sep=",")
I=dim(res)[1]
J=dim(res)[2]
data_irt=list(n_student =I,n_item=J,Y=res)
irt_1pl = Stan(file = irt_1pl.Stan, data = data_irt, iter = 1000, chains = 3)
irt_2pl = Stan(file = irt_2pl.Stan, data = data_irt, iter = 1000, chains = 3)
irt_3pl = Stan(file = irt_3pl.Stan, data = data_irt, iter = 1000, chains = 3)
print(irt_3pl,par="alpha")
traceplot(irt_3pl, pars= "gamma",inc_warmup = FALSE)
log_lik1 = extract_log_lik(irt_1pl)
loo1 = loo(log_lik1)
waic1 = waic(log_lik1)
log_lik2 = extract_log_lik(irt_2pl11)
loo2 = loo(log_lik2)
waic2 = waic(log_lik2)
log_lik3 = extract_log_lik(irt_3pl)
loo3 = loo(log_lik3)
waic3 = waic(log_lik3)
```

Listing 5: R Code for Example 2, the Rasch Testlet Model

```
setwd("C:\\ Rstan Tutorial")
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
data<-read.table("example2.csv",header=FALSE,sep=",")
res<-t(data)
J<-dim(res)[1]
I<-dim(res)[2]
n_tl<-6
t_index<-rep(1:6, each = 6)
data_testlet<-list(n_student = I,n_item=J,n_tl=n_tl,t_index=t_index,res=res)
code_testlet<-'data {
int<lower=0> n_student;
int<lower=0> n_item;
int <lower=0> n_tl; // number of testlets
int <lower=0> t_index[n_item]; // testlet membership
int<lower=0,upper=1> res[n_item,n_student];
}
parameters {
vector[n_student] theta;
```

(continued)

Listing 5: (continued)

```
vector[n_student] gamma[n_tl]; // testlet effect
vector[n_item] beta_free; //freely estimated item difficulty
real<lower=0 > sigma_beta; // sd of difficulty
real<lower=0 > sigma_gamma[n_tl]; // sd of testlet effect
real mu; //mean difficulty
}
transformed parameters {
vector[n_item] beta;
for (i in 1:(n_item-1)) beta[i]=beta_free[i];
beta[n_item]=-1*sum(beta_free); //constraint difficulty sum to 0
}
model {
beta_free ~ normal(0,1);
theta ~ normal(mu,sigma_theta);
mu ~ normal(0,5);
sigma_theta ~ cauchy(0,5);
for (i in 1: n_tl){
gamma[i] ~ normal(0,sigma_gamma[i]);
sigma_gamma[i] ~ cauchy(0,5);
}
for(i in 1:n_item) {
res[i] ~ bernoulli_logit(theta + gamma[t_index[i]]- beta[i]);
}
}
'
irt_testlet <- Stan(model_code = code_testlet, data = data_testlet, iter = 1000, chains = 3)
```

Listing 6: R Code for Example 3, the Multilevel 3PL IRT Model

```
setwd("C:\\Rstan Tutorial")
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
data<-read.table("example3.csv",header=TRUE,sep=",")
res<-data[,-43]
I<-dim(res)[1]
J<-dim(res)[2]
group<-data[,43]
ng<-length(unique(group))
data_m3pl<-list(n_student =I,n_item=J,res=res,g=group,n_school=ng)
code_m3pl<-'data {
int<lower=0 > n_student;
int<lower=0 > n_item;
int<lower=0 > n_school;
int<lower=0 > g[n_student];
int<lower=0,upper=1 > res[n_student,n_item];
}
```

(continued)

Listing 6: (continued)

```
parameters {
vector[n_student] theta;
vector[n_school] s_theta; //program mean ability
vector[n_item] beta;
vector <lower=0 > [n_item] alpha;
vector <lower=0,upper=1 > [n_item] guessing;
real mu;
real<lower=0 > sigma_alpha;
real<lower=0 > sigma_beta;
real<lower=0 > sigma_school; // sd of school mean ability
}
model {
mu ~ normal(0,5);
sigma_beta ~ cauchy(0,5);
beta ~ normal(mu,sigma_beta);
theta ~ normal(0,1);
s_theta ~ normal(0,sigma_school);
alpha ~ lognormal(0,sigma_alpha);
guessing ~ beta(5,23);
sigma_alpha ~ cauchy(0,5);
sigma_school ~ cauchy(0,5);
for(i in 1:n_student) {
for (j in 1:n_item) {
real p;
p= inv_logit(alpha[j]*(s_theta[g[i]] + theta[i]- beta[j]));
res[i,j] ~ bernoulli(guessing[j] + (1-guessing[j])*p);
}}
}
'
irt_m3pl <- Stan(model_code = code_m3pl, data = data_m3pl, iter = 3000, chains = 3)
traceplot(irt_m3pl,pars=c("s_theta","sigma_school"),inc_warmup = FALSE)
print(irt_m3pl,pars=c("s_theta","sigma_school"))
```

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

Notes

1. For a detailed but accessible description of how the momentum variable works in HMC, readers are referred to Gelman et al. (2014).
2. Because of space limitation and for better readability, we do not provide Stan codes for all the common IRT models in this article. Interested readers may contact the first author for additional Stan codes for other IRT models.

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*, 716-723.

Ames, A. J., & Samonte, K. (2015). Using SAS PROC MCMC for item response theory models. *Educational and Psychological Measurement*, *75*, 585-609.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika*, *43*, 561-573.

Bock, R. D. (1972). Estimating item parameters and latent ability when responses are scored in two or more nominal categories. *Psychometrika*, *37*, 29-51.

Bolt, D. M., Cohen, A. S., & Wollack, J. A. (2001). A mixture item response model for multiple-choice data. *Journal of Educational and Behavioral Statistics*, *26*, 381-409.

Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., & . . . Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*, 1-32.

Cowles, M. K., & Carlin, B. P. (1996). Markov chain Monte Carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, *91*, 883-904.

Curtis, S. M. (2010). BUGS code for item response theory. *Journal of Statistical Software*, *36*, 1-34.

Fox, J. P., & Glas, C. A. (2001). Bayesian estimation of a multilevel IRT model using Gibbs sampling. *Psychometrika*, *66*, 271-288.

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2014). *Bayesian data analysis*. Boca Raton, FL: Chapman & Hall/CRC Press.

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, *7*, 457-472.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6*, 721-741.

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.

Lunn, D., Jackson, C., Best, N., Thomas, A., & Spiegelhalter, D. (2012). *The BUGS book: A practical introduction to Bayesian analysis*. New York, NY: Chapman &Hall/CRC Press.

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., & Al-Harbi, K. (2016). *Performances of LOO and WAIC as IRT model selection methods*. Paper presented at the International Meeting of Psychometric Society, Ashville, NC.

Martin, A. D., Quinn, K. M., & Park, J. H. (2011). Mcmcpack: Markov chain Monte Carlo in R. *Journal of Statistical Software*, *42*(9), 1-21.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika*, *47*, 149-174.

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.

Neal, R. M. (1993). *Probabilistic inference using Markov chain Monte Carlo methods* (Technical Report CRG-TR-93-1). Toronto, Ontario, Canada: University of Toronto, Department of Computer Science.

Neal, R. M. (2011). MCMC using Hamiltonian dynamics. In S. Brooks, A. Gelman, & X.-L. Meng (Eds.), *Handbook of Markov Chain Monte Carlo* (*Vol. 2*, pp. 113-162). New York, NY: CRC Press.

Plummer, M. (2003, March). JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In K. Hornik, F. Leisch, & A. Zeileis (Eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*. Retrieved from https://www.r-project.org/conferences/DSC-2003/Proceedings/Plummer.pdf

Rusch, T., Maier, M. J., & Hatzinger, R. (2013). Linear logistic models with relaxed assumptions in R. In B. Lausen, D. van den Poel, & A. Ultsch (Eds.), *Algorithms from and for nature and life* (pp. 337-347). New York, NY: Springer.

Samejima, F. (1969). *Estimation of latent ability using a response pattern of graded scores* (Psychometrika Monograph No. 17). Richmond, VA: Psychometric Society.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, *6*, 461-464.

Sinharay, S. (2004). Experiences with Markov chain Monte Carlo convergence assessment in two psychometric examples. *Journal of Educational and Behavioral Statistics*, *29*, 461-488.

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., & Van Der Linde, A. (2002). Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *64*, 583-639.

Spiegelhalter, D. J., Thomas, A., Best, N. G., & Lunn, D. (2010). *OpenBUGS Version 3.1.1 user manual*. Retrieved from http://www.openbugs.net/Manuals/Manual.html

Stan Development Team. (2016a). *ShinyStan: Interactive visual and numerical diagnostics and posterior analysis for Bayesian models* (R package Version 2.2.1). Retrieved from https://cran.r-project.org/web/packages/shinystan/shinystan.pdf

Stan Development Team. (2016b). *Stan modeling language users guide and reference manual* (Version 2.12.0). Retrieved from http://mc-stan.org/documentation/

Stone, C. A., & Zhu, X. (2015). *Bayesian analysis of item response theory models using SAS*. Cary, NC: SAS Institute.

Sturtz, S., Ligges, U., & Gelman, A. (2005). R2WinBUGS: A package for running WinBUGS from R. *Journal of Statistical Software*, *12*(3), 1-16.

Thissen, D., & Steinberg, L. (1986). A taxonomy of item response models. *Psychometrika*, *51*, 567-577.

Vehtari, A., Gelman, A., & Gabry, J. (2016a). *Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC* (arXiv preprint arXiv:1507.04544). Retrieved from https://arxiv.org/abs/1507.04544

Vehtari, A., Gelman, A., & Gabry, J. (2016b). *loo: Efficient leave-one-out cross-validation and WAIC for Bayesian models* (R package Version 0.1.6). Retrieved from https://cran.r-project.org/web/packages/loo/loo.pdf

Wang, W. C., & Wilson, M. (2005). The Rasch testlet model. *Applied Psychological Measurement*, *29*, 126-149.

Watanabe, S. (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research*, *11*, 3571-3594.