

# Fitting Bayesian item response models in Stata and Stan\*

Robert L. Grant<sup>†</sup>, Daniel C. Furr<sup>‡</sup>, Bob Carpenter<sup>§</sup>, and Andrew Gelman<sup>¶</sup>

12 Jan 2015

## Abstract

Stata users have access to two easy-to-use implementations of Bayesian inference: Stata’s native `bayesmh` function and StataStan, which calls the general Bayesian engine Stan. We compare these on two models that are important for education research: the Rasch model and the hierarchical Rasch model. Stan (as called from Stata) fits a more general range of models than can be fit by `bayesmh` and is also more scalable, in that it could easily fit models with at least ten times more parameters than could be fit using Stata’s native Bayesian implementation. In addition, Stan runs between two and ten times faster than `bayesmh` as measured in effective sample size per second: that is, compared to Stan, it takes Stata’s built-in Bayesian engine twice to ten times as long to get inferences with equivalent precision. We attribute Stan’s advantage in flexibility to its general modeling language, and its advantages in scalability and speed to an efficient sampling algorithm: Hamiltonian Monte Carlo using the no-U-turn sampler. In order to further investigate scalability, we also compared to the package Jags, which performed better than Stata’s native Bayesian engine but not as well as StataStan.

Given its advantages in speed, generality, and scalability, and that Stan is open-source and can be run directly from Stata using StataStan, we recommend that Stata users adopt Stan as their Bayesian inference engine of choice.

## 1 Introduction

Stata is a statistical software package that is popular in social science, economics, and biostatistics. In 2015, it became possible to routinely fit Bayesian models in Stata in two different ways: (a) the introduction of Bayesian modeling commands in Stata software version 14 (StataCorp, 2015), which work using the Metropolis-Hastings algorithm and Gibbs sampler; and (b) the release of StataStan, an interface to the open-source Bayesian software Stan (Stan Development Team, 2015). Previously, Bayesian methods were only available in Stata by user-written commands to interface with external software such as BUGS, JAGS, or MLwiN.

At the time of writing, the native Bayes implementation in Stata, `bayesmh`, allows a choice among 10 likelihood functions and 18 prior distributions. The built-in likelihoods are focused around regression models, and extensions to hierarchical (multilevel) models are possible with the inclusion of hyperpriors. In addition, the user may write customized likelihood functions or customized posterior distributions.

Stan is a language for Bayesian inference which allows general continuous-parameter models, including all the models that can be fit in Stata’s `bayesmh` and many others. Stan can run from

---

\*We thank the Institute of Education Sciences for partial support of this work.

<sup>†</sup>Kingston University & St George’s, University of London

<sup>‡</sup>University of California at Berkeley

<sup>§</sup>Columbia University

<sup>¶</sup>Columbia University

various data analysis environments such as Stata, R, Python, and Julia, and also has a command-line interface. Stan uses Hamiltonian Monte Carlo (HMC) and the no-U-turn sampler (NUTS) with also the options of variational inference (Kucukelbir et al., 2015) and the L-BFGS optimization algorithm (Nocedal and Wright, 2006). The advantages of HMC and NUTS in speed, stability, and scalability over Metropolis-Hastings and Gibbs have been described elsewhere (Neal, 2011; Hoffman and Gelman, 2014). As a result of the Hamiltonian dynamics, HMC is rotation-invariant, making it well-suited to highly correlated parameters. It is also not slowed down by non-conjugate models.

The languages used by these packages are notably different. In Stata, each line of code starts with a command (`bayesmh`) and is followed by arguments, typically one or more variable names with options following after a comma. This allows easy specification of certain standard models. However, flexible programming of complex models is difficult with Stata, whereas the more general specification possible in Stan facilitates more flexible model development. Stan works by translating a model into C++ and compiling that code.

In the present paper, we compare Stata’s Bayesian implementation to Stan (as run from Stata) on some item response models. These logistic regression, or Rasch, models, are popular in education research and in political science (where they are called ideal-point models).

## 2 Models

We fit the Rasch (1960) model using data simulated from the model. We check that the Stata and Stan implementations give the same answer (modulo the inevitable Monte Carlo error of these stochastic algorithms) and we then compare the programs on speed and scalability.

The Rasch model can be written as,

$$\Pr(y_{ip} = 1 | \theta_p, \delta_i) = \text{logit}^{-1}(\theta_p + \delta_i) \tag{1}$$

$$\theta_p \sim N(0, \sigma^2), \tag{2}$$

where  $y_{ip} = 1$  if person  $p$  responded to item  $i$  correctly and is 0 otherwise. The parameter  $\theta_p$  represents the latent “ability” of person  $p$ , and  $\delta_i$  is a parameter for item  $i$ . We shall consider a simple version of the model in which the abilities are modeled as exchangeable draws from a normal distribution with scale  $\sigma$ . Ordinarily the model is specified such that  $\delta_i$  is subtracted from  $\theta_p$ , giving  $\delta_i$  the interpretation of item difficulty. We use the above parameterization because the alternative is more difficult to express in Stata’s `bayesmh`. We assign the following prior distributions to  $\delta_i$  and  $\sigma$ :

$$\delta_i \sim N(0, 10) \tag{3}$$

$$\sigma^2 \sim \text{Inv-Gamma}(1, 1). \tag{4}$$

These priors closely match those given in the Rasch model example in the Stata 14 manual StataCorp (2015). In general we do not recommend this sort of inverse-gamma prior, as it can be more informative than users realize (Gelman, 2006), but we use it here to be consistent with Stata’s documentation. It is easy enough in Stan (or StataStan) to switch in other priors.

A natural hierarchical extension of the Rasch model adds a hyperprior for  $\delta_i$  so that,

$$\Pr(y_{ip} = 1 | \theta_p, \delta_i) = \text{logit}^{-1}(\theta_p + \delta_i) \tag{5}$$

$$\theta_p \sim N(0, \sigma^2) \tag{6}$$

$$\delta_i \sim N(\mu, \tau^2), \tag{7}$$

where  $\mu$  is the model intercept. Persons and items are regarded as two sets of exchangeable draws. The prior distributions are

$$\mu \sim N(0, 10) \tag{8}$$

$$\tau^2 \sim \text{Inv-Gamma}(1, 1) \tag{9}$$

$$\sigma^2 \sim \text{Inv-Gamma}(1, 1). \tag{10}$$

### 3 Methods

We simulated data from the above model with 500 persons answering 20 items. For true values of  $\delta_i$ , we assigned equally-spaced values from  $-1.5$  to  $1.5$ , and we set the true  $\sigma$  to 1.

We set up the Rasch and hierarchical Rasch models in a similar manner, running four chains in series in Stan version 2.8 and Stata version 14.1. We drew initial values for the chains from independent uniform distributions  $-1$  to  $1$  on the location parameters  $\mu^{(0)}$ ,  $\delta^{(0)}$ , and  $\theta^{(0)}$ ; and uniform distributions from  $0$  to  $2$  on the scale parameters  $\sigma^{(0)}$  and  $\tau^{(0)}$ . We assigned all  $\delta_i$ 's identical starting values for each chain, and the same for the  $\theta_p$ 's. The reason for this (admittedly unusual) choice is that this approach is much easier to employ with `bayesmh`. We used the same starting values for both Stan and Stata (and in the scalability comparison described below, for JAGS). These item-response models were not sensitive to starting values.

We ran ten chains for 2500 discarded warm-up iterations and 2500 posterior draws each. For timing purposes, we ran all chains in serial, thus eliminating one of Stan's advantages which is that it can automatically run multiple chains in parallel on a multi-core machine, regardless of the version of Stata. We provide the Stan programs and Stata function calls in the appendix. The options specified for the Stata function call for the Rasch model are nearly identical to those in the example provided in the Stata manual StataCorp (2015).

We monitored convergence for each parameter using the  $\widehat{R}$  statistic, which is a rough estimate of the square root of the ratio of overall (across chains) posterior variance to within-chain posterior variance (Gelman et al., 2013). Values of  $\widehat{R}$  near 1 imply convergence, while greater values indicate non-convergence. Values less than 1.1 are generally considered acceptable. The efficiency of the estimations is evaluated by the estimated effective sample size per second,  $\hat{n}_{\text{eff}}/s$  (Gelman et al., 2013). The timings throughout excluded compiling time in Stan but included the warm-up iterations; in Stata's `bayesmh` they cover the time that Stata takes to run the command for both warm-up and sampling.

To further investigate the scalability of the software, we carried out the same analyses on simulated data with 20 items and 100, 500, 1000, 5000 and 10,000 people. We compared Stan 2.8 (using StataStan) and Stata 14.1 `bayesmh` as above and also the open-source software JAGS 4.0.0 (Plummer, 2007) via the `rjags` package in R 3.2.2, and ran four chains in each instance.

### 4 Results

For the Rasch model, we ran Stan (StataStan) for ten chains of 5,000 iterations (first half as warm-up) in 54 minutes; at that point,  $\widehat{R}$  was less than 1.01 for all parameters. We ran Stata (`bayesmh`) for ten chains of the same length in 26 minutes;  $\widehat{R}$  was less than 1.02 for all parameters. Convergence appears satisfactory for both. Figure 1 compares values of  $\hat{n}_{\text{eff}}/s$  between Stan and Stata for  $\delta$  and  $\theta$ . Table 1 provides the same information for the remaining parameter,  $\sigma$ . Stan was also more efficient, with values of  $\hat{n}_{\text{eff}}/s$  between three and seven times those for Stata across all parameters. Our timings include warmup but not compilation time.

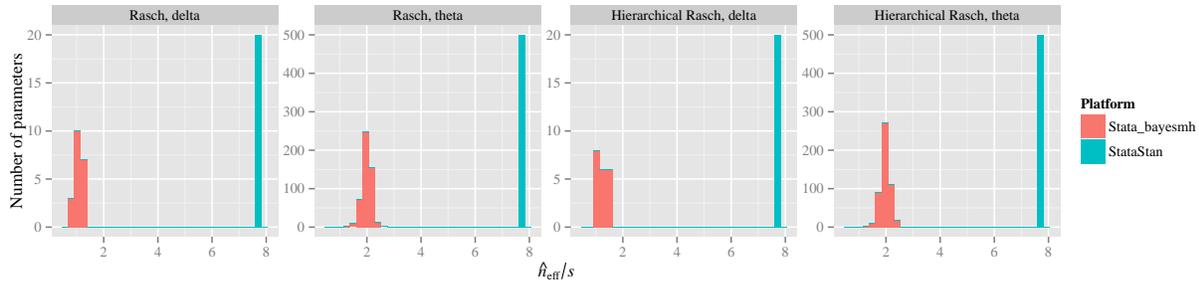


Figure 1: *Histograms of effective sample size per second for parameters in the Rasch and hierarchical Rasch models (20 item-level parameters  $\delta$  and 500 person-level parameters  $\theta$ ). Stan (StataStan) is about four times more efficient than Stata’s `bayesmh` command.*

Model	Parameter	Stata 14.1 bayesmh	StataStan
		$n_{\text{eff}}/\text{sec}$	$n_{\text{eff}}/\text{sec}$
Rasch	$\sigma^2$	0.8	5.4
Hierarchical Rasch	$\mu$	2.3	7.8
Hierarchical Rasch	$\sigma^2$	0.9	5.6
Hierarchical Rasch	$\tau^2$	2.4	7.8

Table 1: *Efficiency statistics for the hyperparameters in the two models. Stan (StataStan) is between three and seven times more efficient than Stata’s native Bayesian engine (`bayesmh`).*

Results for the hierarchical Rasch model parallel those for the Rasch model. Estimation with Stan required 54 minutes for the same number of chains and iterations, and  $\widehat{R}$  was less than 1.01 for all parameters. Stata ran for 26 minutes and yielded values of  $\widehat{R}$  less than 1.01 for all parameters. Both estimations appear to have converged. Stan again was more than three times as efficient as Stata in terms of  $\widehat{n}_{\text{eff}}/s$ .

In scalability testing, all three packages showed an approximately linear relationship between time per effective sample and number of parameters on the logarithmic scale, but Stan was consistently more than twice faster than JAGS, and more than five times faster than Stata. As the size of the data and models became large ( $p > 1000$ ), Stata’s `bayesmh` and JAGS both failed, returning error messages related to memory availability, while StataStan completed up to the desired maximum of  $p = 10,000$ , and presumably could grow beyond this. Observed times per effective sample size are shown in Figures 2 and 3.

## 5 Discussion

The implementation of adaptive Hamiltonian Monte Carlo in Stan is substantially more efficient than the adaptive Markov chain Monte Carlo algorithm featured in Stata for the Rasch and hierarchical Rasch models. We would expect the results to generalize to logistic generalized linear mixed models, given that these include the Rasch models as a special case (Rijmen et al., 2003; Zheng and Rabe-Hesketh, 2007).

The Stata interface to Stan, StataStan, operates by writing specified variables (as vectors), matrices and scalars from Stata to a text file and calls the command-line implementation of Stan. Progress is displayed inside Stata (even under Windows) and there is the option to write the Stan model inside a comment block in the Stata code (which Stata users call a do-file). Results can then

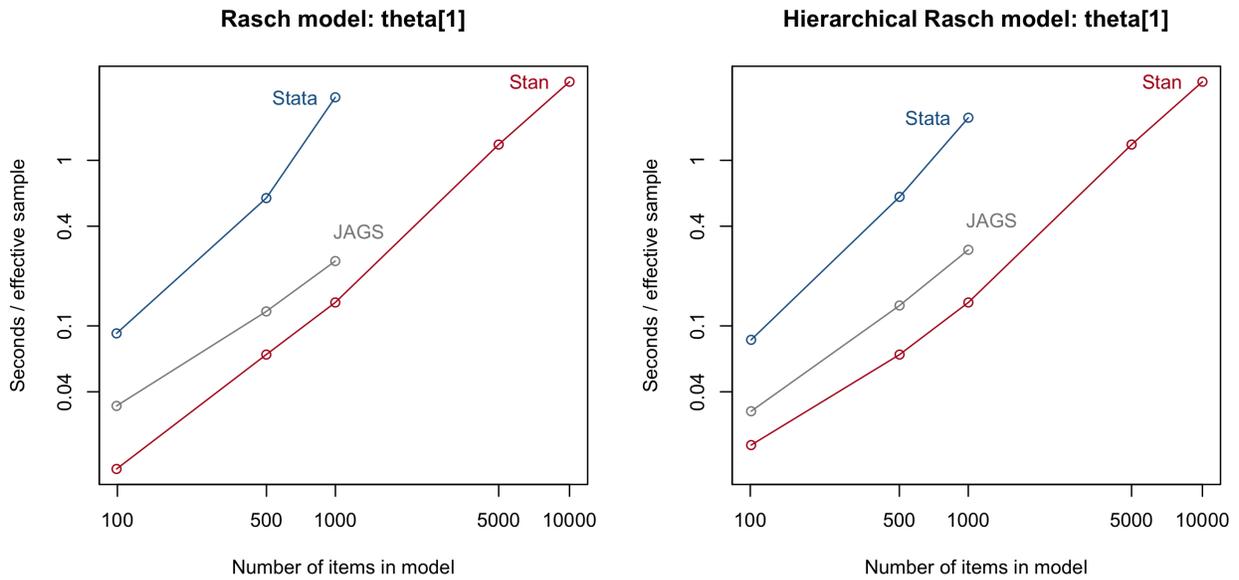


Figure 2: *Time per effective sample size for the first  $\theta$  parameter in the Rasch and hierarchical Rasch models, plotted against the number of  $\theta$  parameters in the model. Stan is faster than the alternatives and is more scalable in the sense of being able to fit larger models to more data.*

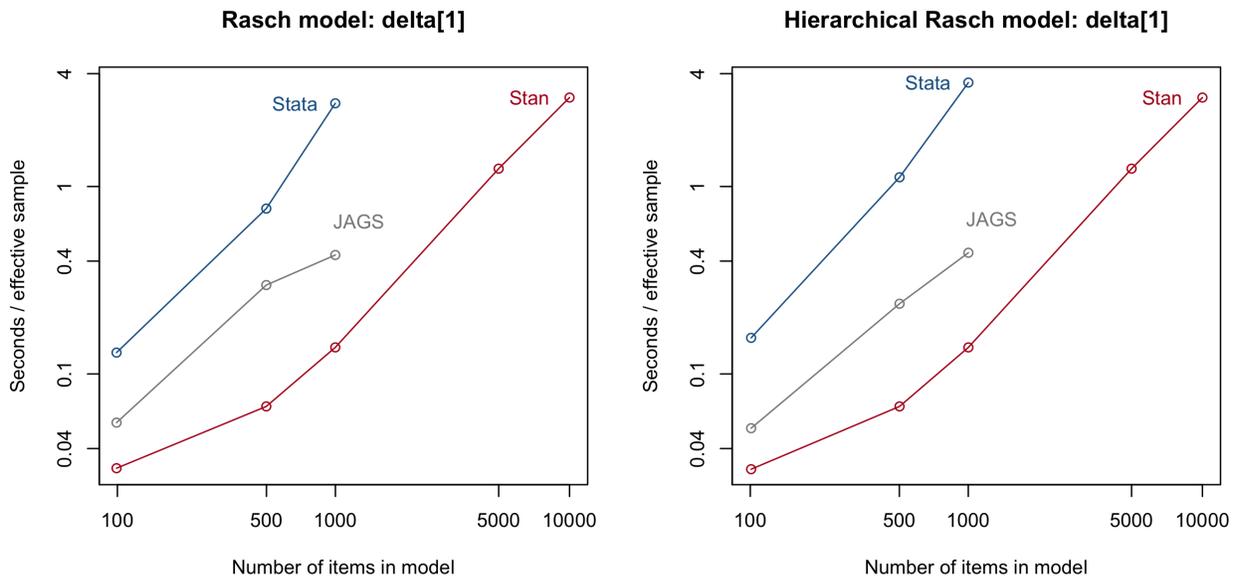


Figure 3: *Time per effective sample size for the first  $\delta$  parameter in the Rasch and hierarchical Rasch models, plotted against the number of  $\theta$  parameters in the model. Stan (here, StataStan) again is consistently faster and scales better to larger problems.*

be read back into Stata for diagnostics and saving in Stata’s data format.

In conclusion, we find Stan to be faster than Stata’s current implementation of Bayesian inference, which is no surprise given Stan’s advanced algorithms and efficient autodifferentiation code. Given that Stan is open-source, and that Stan can be run directly from Stata using StataStan, we recommend that Stata users adopt Stan as their Bayesian inference engine of choice.

## References

- Gelman, A. (2006). Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis* 1, 515–533.
- Gelman, A., J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin (2013). *Bayesian Data Analysis* (third ed.). CRC Press.
- Hoffman, M. and A. Gelman (2014). The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research* 15, 1593–1623.
- Kucukelbir, A., R. Ranganath, A. Gelman, and D. Blei (2015). Automatic variational inference in stan. *ArXiv*, 1506.03431.
- Neal, R. (2011). MCMC using Hamiltonian dynamics. In S. Brooks, A. Gelman, G. Jones, and X.-L. Meng (Eds.), *Handbook of Markov Chain Monte Carlo*. CRC Press.
- Nocedal, J. and S. Wright (2006). *Numerical Optimization* (second ed.). Springer-Verlag.
- Plummer, M. (2007). Jags: Just another Gibbs sampler. <http://mcmc-jags.sourceforge.net/>.
- Rasch, G. (1960). *Probabilistic Models for Some Intelligence and Achievement Tests*. University of Chicago Press.
- Rijmen, F., F. Tuerlinckx, P. De Boeck, and P. Kuppens (2003). A nonlinear mixed model framework for item response theory. *Psychological Methods* 8(2), 185–205.
- Stan Development Team (2015). Stan modeling language: User’s guide and reference manual. <http://mc-stan.org/documentation/>.
- StataCorp (2015). *Stata Statistical Software: Release 14*. StataCorp LP.
- Zheng, X. and S. Rabe-Hesketh (2007). Estimating parameters of dichotomous and ordinal item response models with gllamm. *Stata Journal* 7(3), 313–333.

## Appendix

Here is the code for the Rasch model in Stan. As noted in the text, we would not usually use the inverse-gamma prior distribution on  $\sigma^2$ ; we do so to keep our model comparable to what is documented for Stata.

```
data {  
  int<lower=1> N;  
  int<lower=1> I;  
  int<lower=1> P;
```

```

    int<lower=1, upper=I> ii[N];
    int<lower=1, upper=P> pp[N];
    int<lower=0, upper=1> y[N];
}
parameters {
    real<lower=0> sigmasq;
    vector[I] delta;
    vector[P] theta;
}
model {
    vector[N] eta;
    theta ~ normal(0, sqrt(sigmasq));
    delta ~ normal(0, sqrt(10));
    sigmasq ~ inv_gamma(1, 1);
    for (n in 1:N)
        eta[n] <- theta[pp[n]] + delta[ii[n]];
    y ~ bernoulli_logit(eta);
}

```

Here is the Stata call for the Rasch model:

```

bayesmh y i.item, noconstant reffects(person) likelihood(logit) ///
    mcmcsize(2500) burnin(2500) ///
    prior({y:i.person}, normal(0, {sigmasq})) ///
    prior({y:i.item}, normal(0, 10)) ///
    prior({sigmasq}, igamma(1,1)) ///
    block({sigmasq}) block({y:i.item}, reffects)

```

And here is the JAGS code for the Rasch model:

```

model {
    for (i in 1:I) {
        delta[i] ~ dunif(-1e6, 1e6)
    }
    sigma ~ dunif(0, 1e6)
    inv_sigma_sq <- pow(sigma, -2)
    for (p in 1:P) {
        theta[p] ~ dnorm(0, inv_sigma_sq)
    }
    for (n in 1:N) {
        # eta on different scale here
        logit(inv_logit_eta[n]) <- theta[pp[n]] + delta[ii[n]]
        y[n] ~ dbern(inv_logit_eta[n])
    }
}

```

Here is the hierarchical Rasch model in Stan:

```

data {
    int<lower=1> N;
    int<lower=1> I;
    int<lower=1> P;
    int<lower=1, upper=I> ii[N];
    int<lower=1, upper=P> pp[N];
    int<lower=0, upper=1> y[N];
}

```

```

parameters {
  real<lower=0> sigmasq;
  real<lower=0> tausq;
  real mu;
  vector[I] delta;
  vector[P] theta;
}
model {
  vector[N] eta;
  theta ~ normal(0, sqrt(sigmasq));
  delta ~ normal(mu, sqrt(tausq));
  mu ~ normal(0, sqrt(10));
  sigmasq ~ inv_gamma(1, 1);
  tausq ~ inv_gamma(1, 1);
  for (n in 1:N)
    eta[n] <- theta[pp[n]] + delta[ii[n]];
  y ~ bernoulli_logit(eta);
}

```

Here is the Stata call for the hierarchical Rasch model:

```

bayesmh y i.item, noconstant reffects(person) likelihood(logit) ///
  mcmcsize(2500) burnin(2500) ///
  prior({y:i.person}, normal(0, {sigmasq})) ///
  prior({y:i.item}, normal({mu}, {tausq})) ///
  prior({mu}, normal(0, 10)) ///
  prior({sigmasq} {tausq}, igamma(1,1)) ///
  block({sigmasq} {tausq} {mu}, split) block({y:i.item}, reffects)

```

And here is the JAGS code for the hierarchical Rasch model:

```

model {
  for (i in 1:I) {
    delta[i] ~ dunif(-1e6, 1e6)
  }
  sigma ~ dunif(0, 1e6)
  inv_sigma_sq <- pow(sigma, -2)
  for (p in 1:P) {
    theta[p] ~ dnorm(0, inv_sigma_sq)
  }
  for (n in 1:N) {
    # eta on different scale here
    logit(inv_logit_eta[n]) <- theta[pp[n]] + delta[ii[n]]
    y[n] ~ dbern(inv_logit_eta[n])
  }
}

```