Comparing the MCMC efficiency of JAGS and Stan for the multi-level intercept-only model in the covariance- and mean-based and classic parametrization
Abstract

Bayesian MCMC is a widely used model estimation technique, and software from the BUGS family, such as JAGS, have been popular for over two decades. Recently, Stan entered the market with promises of higher efficiency fueled by advanced and more sophisticated algorithms. With this short study we want to contribute empirical results to the discussion about the sampling efficiency of JAGS and Stan. We conducted three simulation studies in which we varied the number of warmup iterations, the prior informativeness, and sample sizes and employed the multi-level intercept-only model in the covariance- and mean-based and in the classic parametrization. The target outcome was MCMC efficiency measured as effective sample size per second (ESS/s). Based on our specific (and limited) study setup, we found that (1) MCMC efficiency is much higher for the covariance- and mean-based parametrization than for the classic parametrization, (2) Stan clearly outperforms JAGS when the covariance- and mean-based parametrization is used, and that (3) JAGS clearly outperforms Stan when the classic parametrization is used.

Keywords: JAGS, Stan, effective sample size, MCMC efficiency, Bayesian SEM
Comparing the MCMC efficiency of JAGS and Stan for the multi-level intercept-only model in the covariance- and mean-based and classic parametrization.

Bayesian statistics is gaining in popularity in many disciplines and are used for many different purposes, for instance, to include previous knowledge, to estimate otherwise intractable models, to model uncertainty (e.g., Van de Schoot et al., 2017), and to stabilize parameter estimates (e.g., Zitzmann, 2018).

A popular software platform for Bayesian estimation is the BUGS family including BUGS, WinBUGS, OpenBUGS, and JAGS. Monnahan et al. (2017, p. 339) even call BUGS the “workhorse for Bayesian analyses in ecology and other fields for the last 20 years.” More recently, the software Stan whose development was inspired by the “path-breaking programs” BUGS (Lunn et al., 2009, 2013) and JAGS (Gelman et al., 2015, p. 9) and “motivated by the desire to solve problems that could not be solved in a reasonable time [...] using other packages” (p. 8) entered the market with promises of higher computational and algorithmic efficiency. Often, the superiority of Stan over JAGS, which is a more modern member of the BUGS family, is claimed to be due to more advanced MCMC algorithms. Whereas JAGS uses conjugate and slice sampling, Stan uses the No-U-Turn Sampler (NUTS; Hoffman & Gelman, 2014), which is an adaptive variant of Hamiltonian Monte Carlo (HMC; Neal, 2011). A comprehensive illustration of NUTS and HMC can be found in the work of Monnahan et al. (2017) and more detailed technical descriptions in the works of Nishio and Arakaw (2019) and Betancourt (2018).

There has been much debate on efficiency differences between Stan and JAGS, and some authors have explored this research question by conducting comparison studies. For instance, Carpenter et al. (2017, p. 10) found that “Compared to BUGS and JAGS, Stan is often relatively slow per iteration but relatively fast to generate a target effective sample size.” Monnahan et al. (2017, p. 339) conclude that “[f]or small, simple models there is little practical difference between the two platforms, but Stan outperforms BUGS as model size and complexity grows.” which is in line with Gelman et al.’s (2015, p. 9) state-
ment that “Stan is faster for complex models and scales better than Bugs or Jags for large data sets”. However, Grant et al. (2017) compared the performance (total time per effective sample size) of various software (including StataStan and JAGS) depending on the number of parameters in the Rasch and hierarchical Rasch model and found that “no one software option dominated” (p. 350). Also, Merkle et al. (2020, p. 2) report that the original Stan implementation in package blavaan “was not much faster or more efficient than the JAGS approach.”, and Wingfeet (2014) concluded that “[n]either JAGS nor Stan came out clearly on top”. Further, Bølstad (2019) reports mixed results depending on the conjugacy of the priors, with JAGS beating Stan for a fully conjugate hierarchical model. For a completely non-conjugate model with t-distributions instead of normal distribution, the effect reversed, with Stan being much faster.

In summary, the competition between JAGS and Stan has not been finally decided and performance might depend on several factors, for instance, on the model itself, its complexity, number of parameters, priors, and the parametrization.

**Purpose and Scope**

The purpose of the present work is to contribute to the discussion about the efficiency of JAGS and Stan. To this end, we conducted three simulation studies in which we varied the number of warmup iterations, the informativeness of the prior distributions, the sample sizes, and the model parametrization, and compared the MCMC efficiency operationalized as the effective sample size per second (ESS/s) between JAGS and Stan. The targeted model was the multi-level intercept-only model, which is a popular model in, for example, psychological research and the building block for many more complex multi-level models.

The article is organized into the following sections. First, we describe our Simulation Study I including the data generation, the simulation design, the analysis approaches and procedures, and the results of our simulations. As suggested by anonymous reviewers, we
extended the scope of our work by adding Simulation Study 2 (in which we explored a small sample scenario) and Simulation Study 3 (in which we used another model parametrization). Second, we conclude with a discussion of our work. Annotated JAGS/rjags and Stan/rstan code and an example generated data file are provided in the supplementary material and Appendices A–G.

**Simulation Study 1**

In this simulation study, the MCMC efficiency of estimating the covariance- and mean-based parametrization of the multi-level intercept-only with JAGS and Stan is explored.

**Data Generation**

The data generating model was the multi-level intercept-only model with overall mean $\mu = 0$, level 2 variance $\sigma_\theta^2 = 1$, residual variance $\sigma_\epsilon^2 = 1$, $J = 1,000$ level 2 units, and $P = 20$ level 1 units:

$$y_{jp} \sim N(\theta_j, \sigma_\epsilon^2), \quad (1)$$

$$\theta_j \sim N(\mu, \sigma_\theta^2), \quad (2)$$

where $y_{jp}$ is the $p$th value of level 2 unit $j$, and $\theta_j$ is unit $j$’s mean parameter. The number of generated data sets (replications) was $N_{\text{repl}} = 1,000$. Each of these data sets were analyzed within all 16 design cells of the simulation design (described below).

**Simulation Design**

We varied the following factors in our simulation study: software (JAGS, Stan), number of warmup iterations (150, 1,000), and prior informativeness (ordered categories A [lower informativeness], B, C, D [higher informativeness]). These factors were fully crossed, yielding 16 design cells. As priors, we used an inverse gamma distribution $IG(\alpha, \beta)$ with shape $\alpha$ and scale $\beta$ for the variance parameters ($\sigma_\theta^2$, $\sigma_\epsilon^2$) and a normal distribution $N(0, \sigma_\mu^2)$ with variance $\sigma_\mu^2$ for the mean $\mu$. The levels of the prior informativeness factor refer to dif-
fering degrees of prior informativeness: A: \( \alpha = \beta = 0.001, \sigma_p^2 = 10,000 \); B: \( \alpha = \beta = 0.01, \sigma_p^2 = 2500 \); C: \( \alpha = \beta = 0.1, \sigma_p^2 = 100 \); D: \( \alpha = \beta = 1, \sigma_p^2 = 25 \). Thus, the prior informativeness ranges from lower (A) to higher (D).

**Analysis**

The simulation study was conducted with the statistical software R (R Core Team, 2021). For JAGS (Plummer, 2017), the R package rjags (Plummer, 2019), and for Stan (Stan Development Team, 2021a), the R package rstan (Stan Development Team, 2020) was used. The analysis model was similar to the data generating model, but we used the covariance- and mean-based implementation of the multi-level intercept-only model (Hecht et al., 2020):

\[
S \sim \mathcal{W}_p(\Sigma, J - 1),
\]

\[
\bar{y} \sim \mathcal{N}_p(\mu, \frac{1}{J} \Sigma),
\]

where \( S \) is the sample scatter matrix, \( \bar{y} \) is the sample mean vector, \( \mathcal{W} \) is the Wishart distribution, and where \( \Sigma \) and \( \mu \) are the model-implied covariance matrix and mean vector:

\[
\Sigma = \begin{pmatrix}
\sigma^2 + \sigma^2_e & \sigma^2_e \\
\sigma^2_e & \sigma^2_e + \sigma^2 \\
\end{pmatrix},
\]

\[
\mu = \begin{pmatrix}
\mu_1 \\ \vdots \\ \mu_p
\end{pmatrix}.
\]

The inverse of \( \Sigma \) is:

\[
C = \Sigma^{-1} = \begin{pmatrix}
\frac{(P-1)\sigma^2 + \sigma^2_e}{\xi} & -\frac{\sigma^2}{\xi} \\
-\frac{\sigma^2}{\xi} & \frac{(P-1)\sigma^2 + \sigma^2_e}{\xi}
\end{pmatrix},
\]

with \( \xi = P\sigma^2 \sigma^2_e + \sigma^2_e \).
This “covariance- and mean-based approach” (Hecht et al., 2020) is conceptually similar to the “marginal Stan method” (Merkle et al., 2020) as both groups of authors capitalize on the idea of integrating latent variables out of the model likelihood and has, for instance, been shown to be beneficial for MCMC estimation of continuous-time models (Hecht & Zitzmann, 2020). The approach of formulating models for continuous variables in terms of multivariate normal and Wishart distributions has also previously been described in, for instance, the work of Goldstein (2011).

The parametrization of the Wishart distribution differs in JAGS and Stan. Whereas JAGS’s \texttt{dwish(\Sigma^{-1}, df)} function uses the inverse covariance matrix, Stan’s \texttt{wishart(df, \Sigma)} function uses the covariance matrix. To make the comparison between JAGS and Stan as fair as possible, we avoided time-consuming in-software matrix inversion by passing the parametrization-consistent matrix to the functions. Avoiding the inverse of large covariance matrices has also been recommended by Goldstein (2011).

Each replication ran on one Intel Xeon E5–2687W (3.0 GHz) CPU of a 64-bit Windows Server 2008 system with a total of 48 CPUs, on which we ran 36 replications (each on one core) in parallel. Run times for the following process steps were obtained. For JAGS, the warmup run time \texttt{[jags.model()]} and the sampling run time \texttt{[jags.samples()]} were determined by the before-after difference of time stamps obtained from the function \texttt{Sys.time()}. For Stan, the warmup and sampling run times can be obtained from the console output of the function \texttt{sampling()} and are also retrievable from the returned results object \texttt{(attributes(fitted@sim$samples[[1]])[['elapsed_time']])}. Additionally, we recorded the time to set up and compile the Stan model \texttt{(stan_model(), stanc())} and also the run time of the function \texttt{sampling()} via \texttt{Sys.time()} differences. We report all run times in seconds. The number of warmup iterations [set via argument \texttt{n.adapt} in \texttt{jags.model()} and argument \texttt{warmup} in Stan’s \texttt{sampling()}] was varied in the simulation design (see above), whereas the number of sampling iterations [set via argument \texttt{n.iter} in \texttt{jags.samples()} and argument \texttt{iter} in Stan’s \texttt{sampling()}] was 100,000. One chain per parameter was used.
without thinning. Starting values were the true parameter values (see Data Generation
above).

The effective sample size (ESS) and the potential scale reduction (PSR) were com-
puted with R package shinystan (Gabry, 2018) for both JAGS and Stan samples. The
mode of the converged chain served as the parameter estimate. As parameter recovery
and precision statistics, bias, root mean squared error (RMSE), and the 95% coverage rate
were computed. Our target outcome variable is the MCMC efficiency, which we calculated
as the ratio of the ESS and the run time of the sampling on one CPU. Carpenter et al.
(2017, p. 10) termed this ESS/s (or its inverse) the “most relevant statistic for comparing
the efficiency of sampler implementations” because the estimation accuracy is governed by
the square root of the ESS, a fact that has also been shown by Zitzmann and Hecht (2019;
see also Zitzmann et al., 2021).

Results

Descriptive statistics are presented in Table 1. The maximum PSR value is 1.0002
(JAGS) and 1.0003 (Stan), respectively, indicating that all chains had converged. Visual
inspection of randomly selected trace plots confirmed trouble-free convergence. Mean ESS
values were 74,334 for JAGS and 86,486 for Stan. Hence, Stan produced a higher ESS
than JAGS on average.

Average bias is very close to zero ($M_{bias,JAGS} = -0.0009$, $M_{bias,Stan} = 0.0005$), RMSEs
are practically equal ($M_{RMSE,JAGS} = 0.0297$, $M_{RMSE,Stan} = 0.0303$), and the average coverage
rates nearly hit 0.95 ($M_{CR,JAGS} = 0.9608$, $M_{CR,Stan} = 0.9520$). Bias, RMSE, and coverage
rates of JAGS and Stan are very comparable. Thus, both software estimate the parame-
ters of the multi-level intercept-only model similarly well.

Stan needed on average 0.10s to set up the model [stan_model()] and 56.83s for
compilation [stanc()]. For the warmup, Stan [console output from rstan’s function sampling()]
took 0.36s on average, whereas JAGS [rjags’s function jags.model()] needed 0.87s on av-
average (however, time to set up the model was included in this warmup run time for JAGS, but not for Stan). For the sampling of the 100,000 values, JAGS [\texttt{jags.samples()}] needed 114.99 s and Stan [console output from rstan\textquotesingle s function \texttt{sampling()}] 52.23 s. Thus, Stan was about twice as fast as JAGS. Considering that a higher ESS was achieved in much shorter time, Stan clearly samples more efficiently than JAGS on average (in our simulation setup). However, in addition to warmup and sampling, Stan\textquotesingle s \texttt{sampling()} function took another 215 seconds on average before returning the samples. Thus, users need to wait much longer for the results of the sampling process when using rstan instead of rjags.

Figure 1 shows the MCMC efficiency (i.e., the effective sample size produced in one second) split by the levels of the simulation factors and the model parameters. Additional to these marginal mean MCMC efficiencies, we present mean MCMC efficiencies for the simulation factors and model parameters split by software JAGS and Stan in Figure 2 to investigate interaction effects. The overall mean MCMC efficiency was 1180 ESS/s. Considering software, the average MCMC efficiency is 649 for JAGS and 1712 for Stan. Thus, Stan outperforms JAGS by roughly a factor of two and a half on average. Mean MCMC efficiency is lower for 150 warmup iterations ($M_{\text{warmup}=150} = 1034$) than for 1000 warmup iterations ($M_{\text{warmup}=1000} = 1327$). From Figure 2 it becomes clear that there is an interaction between software and number of warmup iterations. Whereas JAGS\textquotesingle s MCMC efficiency is approximately equal for 150 and 1000 warmup iterations ($M_{\text{warmup}=150,\text{JAGS}} = 652, M_{\text{warmup}=1000,\text{JAGS}} = 647$), Stan profits very much from more warmup iterations ($M_{\text{warmup}=150,\text{Stan}} = 1416, M_{\text{warmup}=1000,\text{Stan}} = 2007$). The prior informativeness has practically no effect on the MCMC efficiency ($M_A = 1180, M_B = 1175, M_C = 1181, M_D = 1186$).

With respect to the three model parameters, the MCMC efficiency differs on average ($M_\mu = 1243, M_{\sigma^2_\theta} = 1053, M_{\sigma^2_\varepsilon} = 1246$), and an interaction with software is evident. Whereas JAGS\textquotesingle s MCMC efficiency for both variance parameters is equal ($M_{\sigma^2_\theta,\text{JAGS}} = 539, M_{\sigma^2_\varepsilon,\text{JAGS}} = 540$), it is higher for the mean ($M_\mu,\text{JAGS} = 869$). For Stan, the picture is different. Here, $\mu$ and $\sigma^2_\theta$ show less MCMC efficiency ($M_\mu,\text{Stan} = 1616, M_{\sigma^2_\theta,\text{Stan}} = 1567$) than the residual variance...
Effect sizes $\eta^2$ for the simulation factors are presented in Table 2. Software exhibits the by far highest variance explanation (68.4%). Warmup iterations and Parameter explain 5.2% and 2.0% of the variance in MCMC efficiency, respectively, whereas variance explanation by Prior informativeness is essentially zero. Interactions with above zero variance explanation are Software $\times$ Warmup Iterations (5.4%), Software $\times$ Parameter (4.5%), Warmup Iterations $\times$ Parameter (1.0%), and Software $\times$ Warmup Iterations $\times$ Parameter (1.0%).

In summary, both software estimate the multi-level intercept-only model equally well, but Stan outperforms JAGS in the production of effective sample size per time unit. Further, Stan profits from more warmup iterations, the prior informativeness is practically not related to the MCMC efficiency, and the MCMC efficiency differs between model parameters.

**Simulation Study 2: Small Sample Size**

In this simulation study, the MCMC efficiency of estimating the covariance- and mean-based parametrization of the multi-level intercept-only model with JAGS and Stan is explored for a small sample size scenario. The simulation design and the analysis strategy were similar to Simulation Study 1. The data generation was similar as well, except that the number of level 2 units was reduced to $J = 100$ and the number of level 1 units to $P = 5$.

The overall mean efficiency in this small sample scenario is 8343 ESS/s and thus roughly seven times higher than in Simulation Study 1 where sample sizes were larger ($J = 1,000$, $P = 20$). Investigating the mean MCMC efficiencies split by the simulation factors, model parameters, and software (Figure 3) yields basically the same pattern as in Simulation Study 1. Stan outperforms JAGS; however, JAGS’s underperformance is not as pronounced as in the large sample scenario. Concerning warmup, the figure again shows that JAGS does not profit from more warmup iterations, whereas Stan does. Prior
informative exhibits no clear effect. An interaction of parameter and software can again be identified. Whereas JAGS performs better in efficiently estimating $\mu$ than $\sigma_0^2$ and $\sigma_1^2$, Stan is approximately equally efficient in estimating all three model parameters.

In summary, reducing the sample size resulted in higher overall MCMC efficiency, but software differences in MCMC efficiency remained, although JAGS caught up somewhat to Stan.

**Simulation Study 3: Classic Parametrization**

In this simulation study, the MCMC efficiency of estimating the classic parametrization of the multi-level intercept-only model with JAGS and Stan is explored. The data generation and the simulation design were similar to Simulation Study 1. For the analysis model, the classic parametrization was now chosen, in which the parameters of the level 2 units ($\theta$, in Equations 1 and 2) were part of the model formulation and thus needed to be sampled (see Hecht et al., 2020, for further details on the differences between the classic and the covariance- and mean-based parametrization of the multi-level intercept-only model).

With $J=1,000$ level 2 units, this meant that 1,000 additional parameters needed to be sampled (an increase by a factor of 334 compared to Simulation Study 1). Hence, to keep the simulation in manageable boundaries, the number of sampling iterations was reduced to 10,000.

Compared to Simulation Study 1 and 2 (with the covariance- and mean-based parametrization), two major differences emerged: (1) The overall mean efficiency (137 ESS/s) is by far lower, and (2) the software rank order reversed: When employing the classic parametrization, JAGS is much more efficient than Stan (see Figure 4). Again, Stan profits from more warmup iterations, whereas JAGS does not, and prior informativeness has no effect on MCMC efficiency. Concerning the parameters, there is no clear differential picture (i.e., within software, the parameters are estimated with roughly the same efficiency), although Stan seems to have problems with estimating the residual variance $\sigma^2_e$ (with just 6
In summary, choosing the classic instead of the covariance- and mean-based implementation resulted in lower overall MCMC efficiency and JAGS clearly outperformed Stan.

Discussion

With this short study we want to contribute empirical results to the discussion about the sampling efficiency of JAGS and Stan. In our limited simulations, Stan outperformed (i.e., exhibited a higher ESS/s) JAGS for the covariance- and mean-based parametrization of the multi-level intercept-only model. However, when the classic parametrization was chosen, JAGS outperformed Stan. Additionally, we found that Stan profited from more warmup iterations and that prior informativeness had no effect on the MCMC efficiency.

The results from our simulation study most certainly will not generalize to other models (or model parametrizations), conditions, or other values/levels of our simulation factors.

Our model was a very simple one. Stan is often said to gain in efficiency with increasing model complexity (e.g., Gelman et al., 2015; Monnahan et al., 2017). We found higher efficiency (roughly by a factor of $2^{1/2}$) already for a simple model. It would be interesting whether Stan outperforms JAGS even more for more complex multi-level models and other relevant models for psychological research.

Besides the model itself, the parametrization of the model is crucial as well. We showed that with the classic parametrization of the multi-level intercept-only model, JAGS clearly outperformed Stan. As the classic parametrization is presumably most intuitive and easiest to implement for psychologists, the software recommendation clearly leans towards JAGS here. However, as shown, users can speed up their analyses (i.e., more efficiently estimate the model parameters) by switching to the covariance- and mean-based
parametrization (a comprehensive tutorial on how to set up this parametrization is given by Hecht et al., 2020). For this parametrization, Stan clearly outperforms JAGS and should be the software of choice (if MCMC efficiency is the target criterion). Our results are in line with Merkle et al.’s (2020) results who reported superiority of their “new Stan approach” which is conceptually similar to Hecht et al.’s (2020) covariance- and mean-based approach. The efficiency of this new approach was even so convincing that the authors of R package blavaan made it their new default method (Merkle et al., 2020, p. 12). However, in contrast to Hecht et al. (2020, Eq. 13 and 14), Merkle et al. (2020, Eq. 5 and 6) use multivariate normal marginal distributions of the sample value vectors. Future research could investigate performance differences between the Wishart and the multivariate normal parametrization. From our experience with JAGS (not published), multivariate normal modeling of the sample value vectors was much slower than the Wishart modeling of the sample scatter matrix, but this might be different for Stan. Future research could compare more parametrizations of multi-level models.

Concerning sample sizes, other studies (e.g., Monnahan et al., 2017) report that Stan’s relative efficiency over JAGS increases with increasing sample size. Our results are in line with this finding (for the covariance- and mean-based parametrization); for our small sample size scenario, Stan did not outperform JAGS as much as in the large sample size scenario. As we only had two sample size scenarios (and varied sample size only for the covariance- and mean-based parametrization), generalizations are limited. Future research could investigate the dependency of sample sizes on the efficiency and the moderating effect of sample size on the difference in efficiency between both software in more detail, especially also with respect to the model parametrization. Also, the relative performance of different methods might depend on the ratio of the variance components in the simulation model.

We used a high number of iterations to achieve sensibly sized absolute runtimes and to obtain reliable ESS estimates (Paganin et al., 2021). We assume that the MCMC efficiency (ESS/s) is constant over the course of sampling. Hence, given the chain has con-
verged, the length of the sampling should have no effect on the MCMC efficiency.

We limited our study to two popular Bayesian software in psychological research, namely JAGS and Stan. Other Bayesian software packages, for instance, NIMBLE (NIMBLE Development Team, 2021), PyMC3 (Salvatier et al., 2016), and LaplacesDemon (Statisticat, 2021), exist and have been the focus of research (e.g., Beraha et al., 2021; de Valpine, 2021; Paganin et al., 2021; Ponisio et al., 2020). Future software comparisons should take these packages even more strongly into account.

The effect of the warmup iterations on Stan’s MCMC efficiency is in line with the functioning of the NUTS algorithm as this algorithm needs to tune the step size to achieve a target acceptance rate and to tune the mass matrix whose function is to transform the posterior to have a simpler geometry for sampling (e.g., Monnahan et al., 2017). As the optimization of the step size and the mass matrix are mutually dependent, a sufficiently long warmup is needed (Monnahan et al., 2017). We had only two warmup iteration sizes (150 and 1,000), with 150 being the lowest default size for one warmup cycle (Stan Development Team, 2021b, p.150). An anonymous reviewer pointed out that the marginal variances of the parameters in our model are rather small; therefore, it is not surprising that one warmup cycle was not sufficient for optimization. In future research, it would be interesting to explore the shape of the functional dependency of the MCMC efficiency on the number of warmup iterations and determine areas of diminishing marginal utility to derive rule-of-thumb thresholds for sufficient warmup of the NUTS algorithm. Additionally, one could facilitate warmup optimization by programming all parameters “so that they have unit scale and so that posterior correlation is reduced; […]” For Hamiltonian Monte Carlo, this implies a unit mass matrix, which requires no adaptation as it is where the algorithm initializes.” (Stan Development Team, 2021c, p.266). Further, the shown warmup dependency should be taken into consideration when comparing results from studies that differed in this aspect. Merkle et al. (2020, p. 8)—who used 300 warmup iterations for Stan—already pointed to this problem and concluded that therefore “the ESS/s metric
We found no effect of the prior informativeness on the MCMC efficiency. However, this result is just valid for the specific four variations of prior informativeness (and all other specifications) in our simulation setup and might not generalize. In fact, in simulation runs with much lower informativeness (not reported), Stan’s MCMC efficiency was lower. Future research could investigate the prior informativeness effect on efficiency with a wider range of informativeness. Also, in our simulation, the amount of data was rather high, marginalizing the effect of the priors. In scenarios with less data, prior effects might arise.

Although Stan was clearly more efficient than JAGS in the sampling phase, the total time users encounter until samples are returned depends on additional steps. Stan models need to be compiled into C++ code prior to sampling, which was considerable in our study (on average, model compilation took longer than the sampling of 100,000 iterations). Of course, once compiled, models may be reused to avoid the extensive compilation time. But still, users who run a model for the first time (or are not aware that once compiled models may be reused) must afford the compilation time, and rstan’s primary user-level function stan() that includes all processes may mask the opportunity for reusing compiled models and contribute to a lengthy user experience. Further, we encountered a relatively huge consumption of additional time besides warmup and sampling by rstan’s sampling() function. In fact, the additional time was about twice the time for all other steps combined. We are not sure what this function needs this additional time for. According to an anonymous reviewer, the additional time is partly due to calculating ESS and PSR. As this needs to be done for JAGS as well, fair software comparisons would also need to take this into account. Maybe future versions of rstan’s sampling() function may include an option to return the samples directly after sampling or explicitly report the time needed for additional calculations of convergence and precision statistics.

To conclude, in our specific study setup, Stan clearly outperformed JAGS in MCMC
efficiency measured as the effective sample size per second for the multi-level intercept-only model.
References

Beraha, M., Falco, D., & Guglielmi, A. (2021). *JAGS, NIMBLE, Stan: A detailed com-
parison among Bayesian MCMC software*. http://arxiv.org/abs/2107.09357

https://arxiv.org/abs/1701.02434

Bølstad, J. (2019, January 2). *How Efficient is Stan Compared to JAGS? Conjugacy,
Pooling, Centering, and Posterior Correlations*. www.boelstad.net/post/stan_vs_jags_speed

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker,

de Valpine (2021, August 8). *Some comparisons between NIMBLE, JAGS and Stan for a
couple of examples from Gelman and Hill (2007)*. https://nature.berkeley.edu/~pdevalpine/
MCMC_comparisons/some_ARM_comparisons/nimble_ARM_comparisons.html

Gabry, J. (2018). *shinystan: Interactive visual and numerical diagnostics and posterior
analysis for Bayesian models (Version 2.5.0) [Software]*. http://cran.r-project.org/
package=shinystan

Bayesian inference and optimization. *Journal of Educational and Behavioral Statis-
tics, 40*, 530–543. https://doi.org/10.3102/1076998615606113


Grant, R. L., Furr, D. C., Carpenter, B., & Gelman, A. (2017). Fitting Bayesian item re-

Hecht, M., Gische, C., Vogel, D., & Zitzmann, S. (2020). Integrating out nuisance param-
eters for computationally more efficient Bayesian estimation – An illustration and
https://doi.org/10.1080/10705511.2019.1647432


NIMBLE Development Team (2021). NIMBLE [Software]. https://r-nimble.org


Plummer, M. (2019). rjags: Bayesian graphical models using MCMC (Version 4-10) [Software]. https://cran.r-project.org/package=rjags


Statisticat, LLC. (2021). LaplacesDemon: Complete Environment for Bayesian Inference [Software].


Table 1

Descriptive Statistics of Convergence, Precision, Recovery, Accuracy, and Run Time by Software (Simulation Study 1)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Software</th>
<th>M</th>
<th>SD</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Convergence/precision</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSR</td>
<td>JAGS</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
<td>1.0002</td>
</tr>
<tr>
<td></td>
<td>Stan</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
<td>1.0003</td>
</tr>
<tr>
<td>ESS</td>
<td>JAGS</td>
<td>74334</td>
<td>17950</td>
<td>46727</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td>Stan</td>
<td>86486</td>
<td>17591</td>
<td>27583</td>
<td>100000</td>
</tr>
<tr>
<td><strong>Recovery/accuracy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bias</td>
<td>JAGS</td>
<td>−0.0009</td>
<td>0.0016</td>
<td>−0.0030</td>
<td>0.0010</td>
</tr>
<tr>
<td></td>
<td>Stan</td>
<td>0.0005</td>
<td>0.0005</td>
<td>−0.0002</td>
<td>0.0010</td>
</tr>
<tr>
<td>RMSE</td>
<td>JAGS</td>
<td>0.0297</td>
<td>0.0155</td>
<td>0.0101</td>
<td>0.0474</td>
</tr>
<tr>
<td></td>
<td>Stan</td>
<td>0.0303</td>
<td>0.0164</td>
<td>0.0096</td>
<td>0.0489</td>
</tr>
<tr>
<td>Coverage rate 95%</td>
<td>JAGS</td>
<td>0.9608</td>
<td>0.0054</td>
<td>0.9530</td>
<td>0.9670</td>
</tr>
<tr>
<td></td>
<td>Stan</td>
<td>0.9520</td>
<td>0.0075</td>
<td>0.9420</td>
<td>0.9620</td>
</tr>
<tr>
<td><strong>Run time (s)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warmup</td>
<td>JAGS</td>
<td>0.87</td>
<td>0.81</td>
<td>0.13</td>
<td>21.95</td>
</tr>
<tr>
<td></td>
<td>Stan</td>
<td>0.36</td>
<td>0.24</td>
<td>0.08</td>
<td>2.05</td>
</tr>
<tr>
<td>Sampling</td>
<td>JAGS</td>
<td>114.99</td>
<td>7.48</td>
<td>89.55</td>
<td>155.42</td>
</tr>
<tr>
<td></td>
<td>Stan</td>
<td>52.23</td>
<td>8.23</td>
<td>30.18</td>
<td>97.65</td>
</tr>
<tr>
<td>Translation</td>
<td>Stan</td>
<td>0.10</td>
<td>0.02</td>
<td>0.06</td>
<td>0.37</td>
</tr>
<tr>
<td>Compilation</td>
<td>Stan</td>
<td>56.83</td>
<td>7.29</td>
<td>42.09</td>
<td>132.18</td>
</tr>
<tr>
<td>sampling()</td>
<td>Stan</td>
<td>214.65</td>
<td>12.86</td>
<td>185.62</td>
<td>270.97</td>
</tr>
</tbody>
</table>

*Note.* Run time for rstan’s `sampling()` function is the additional time that this function runs besides warmup and sampling.
Table 2

*Effect Size $\eta^2$ for the Simulation Factors (Simulation Study 1)*

<table>
<thead>
<tr>
<th>Factor</th>
<th>$\eta^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>0.684</td>
</tr>
<tr>
<td>Warmup Iterations</td>
<td>0.052</td>
</tr>
<tr>
<td>Prior Informativeness</td>
<td>0.000</td>
</tr>
<tr>
<td>Parameter</td>
<td>0.020</td>
</tr>
<tr>
<td>Software × Warmup Iterations</td>
<td>0.054</td>
</tr>
<tr>
<td>Software × Prior Informativeness</td>
<td>0.000</td>
</tr>
<tr>
<td>Software × Parameter</td>
<td>0.045</td>
</tr>
<tr>
<td>Warmup Iterations × Prior Informativeness</td>
<td>0.000</td>
</tr>
<tr>
<td>Warmup Iterations × Parameter</td>
<td>0.010</td>
</tr>
<tr>
<td>Prior Informativeness × Parameter</td>
<td>0.000</td>
</tr>
<tr>
<td>Software × Warmup Iterations × Prior Informativeness</td>
<td>0.000</td>
</tr>
<tr>
<td>Software × Warmup Iterations × Parameter</td>
<td>0.010</td>
</tr>
<tr>
<td>Software × Prior Informativeness × Parameter</td>
<td>0.000</td>
</tr>
<tr>
<td>Warmup Iterations × Prior Informativeness × Parameter</td>
<td>0.000</td>
</tr>
<tr>
<td>Software × Warmup Iterations × Prior Informativeness × Parameter</td>
<td>0.000</td>
</tr>
</tbody>
</table>

*Note.* Dependent variable: MCMC efficiency (ESS/s).
Figure 1. **Simulation Study 1**: ESS performance (ESS/s) of simulation factors software (JAGS/Stan), number of warmup iterations (150/1000), prior informativeness (A/B/C/D), and model parameters ($\mu$, $\sigma_\theta^2$, $\sigma_\epsilon^2$). Number of level 2 units: $J = 1,000$. Number of level 1 units: $P = 20$. Covariance- and mean-based model parametrization.
Figure 2. Simulation Study 1: ESS performance (ESS/s) of number of warmup iterations (150/1000), prior informativeness (A/B/C/D), and model parameters (μ, σ_θ^2, σ_ε^2) by software (JAGS/Stan). Number of level 2 units: J = 1,000. Number of level 1 units: P = 20. Covariance- and mean-based model parametrization.
Figure 3. Simulation Study 2 (Small sample size): ESS performance (ESS/s) of number of warmup iterations (150/1000), prior informativeness (A/B/C/D), and model parameters ($\mu$, $\sigma^2_\theta$, $\sigma^2_\epsilon$) by software (JAGS/Stan). Number of level 2 units: $J = 100$. Number of level 1 units: $P = 5$. Covariance- and mean-based model parametrization.
Figure 4. Simulation Study 3 (Classic parametrization): ESS performance (ESS/s) of number of warmup iterations (150/1000), prior informativeness (A/B/C/D), and model parameters ($\mu$, $\sigma_0^2$, $\sigma^2_\epsilon$) by software (JAGS/Stan). $M_\theta$ is the average MCMC efficiency of level 2 units parameters $\theta_j$. Number of level 2 units: $J = 1,000$. Number of level 1 units: $P = 20$. Classic model parametrization.
Appendix A: rjags Code

# R code for running the multi-level intercept-only model with JAGS/rjags #
###########################################################################
# Supplementary material to article: #
# Hecht, M., Weirich, S., & Zitzmann, S. Comparing the MCMC efficiency of JAGS and Stan #
# for the multi-level intercept-only model #
###########################################################################

## used R version: 4.0.5 (R Core Team, 2021) #
# => get latest version: https://cran.r-project.org/ #

## install JAGS from #
# https://sourceforge.net/projects/mcmc-jags/files #
# used JAGS version: 4.3.0 (Plummer, 2017) #

## load/install packages #
# used rjags version: 4-10 (Plummer, 2019) #
# => get latest rjags version: install.packages( "rjags" ) #
library( rjags )

# used shinystan version: 2.5.0 (Gabry, 2018) #
# => get latest shinystan version: install.packages( "shinystan" ) #
library( shinystan )

## bugs files #
# covariance-/mean-based parametrization #
bugs.file <- c("https://amor.hu-berlin.de/~psymetho/jagsstan/01b_multilevel_intonly_model_covmeanbased.bugs")

# for classic parametrization use: #
# bugs.file <- c("https://amor.hu-berlin.de/~psymetho/jagsstan/01c_multilevel_intonly_model_classic.bugs")

## open bugs file #
bugs.file <- url( bugs.file )

## load simulated example data #
(load(url("https://amor.hu-berlin.de/~psymetho/jagsstan/00_multilevel_intonly_model_example_data.Rdata")))

# D: data matrix in wide format: #
# rows: level 1 units (e.g., measurement occasions) #
# columns: level 2 units (e.g., persons) #
# cells: values
# J: number of level 2 units
# P: number of level 1 units

### preprocessing ###

## compute sample mean vector and sample scatter matrix from data

# sample mean vector y.bar
y.bar <- matrix( rowMeans( D ), P, 1 )

# column vector of ones
ones.vec <- matrix( 1, J, 1 )

# sample scatter matrix
S <- ( D - y.bar %*% t(ones.vec) ) %*% t( ( D - y.bar %*% t(ones.vec) ) )

## data list for JAGS

data.list <- list()
data.list <- c( data.list, list( "S"=S ) )
data.list <- c( data.list, list( "y.bar"=y.bar ) )
data.list <- c( data.list, list( "J"=J ) )
data.list <- c( data.list, list( "P"=P ) )
data.list <- c( data.list, list( "D"=D ) ) # only needed for classic parametrization

## function to generate random starting values

# (starting values are true values from data generation: mu = 0, var.theta = 1, var.eps = 1)
generate.startval <- function(){
  startval <- list()
  startval <- c( startval, list( "mu" = 0 ) )
  startval <- c( startval, list( "prec.theta" = 1/1 ) )
  startval <- c( startval, list( "prec.eps" = 1/1 ) )
  return( startval )
}

# optional for classic parametrization:
# add starting values for parameters of level 2 units (theta)

# set starting values for the parameters (one chain each)
inits <- sapply( 1:1, function (chain.nr,l) l, generate.startval(), simplify=FALSE )

# set seeds for the chain
inits[[1]]$".RNG.name" <- "base::Wichmann-Hill"
inits[[1]]$".RNG.state" <- c( 11076, 3733, 19174 )
### Sampling iterations

```r
# sampling iterations
sampling.iterations <- 100000
```

### Sampling start time

```r
# sampling start time
start.time.sampling <- Sys.time()
```

### Sampling

```r
# sampling
res <- jags.samples( ini, variable.names=c("mu","sigma2.theta","sigma2.eps"),
                      n.iter=sampling.iterations, thin=1, type='trace', progress.bar = NULL )
```

### Optional for classic parametrization:

```r
# add "theta" to argument variable.names
```

### Sampling runtime

```r
# sampling runtime
(runtime.sampling <- Sys.time() - start.time.sampling )
```
# set time units of sampling runtime to seconds (needed below to compute ESS/s)
units( runtime.sampling ) <- "secs"

### postprocessing ###

# create shinystan object
samples <- cbind( matrix( res$mu[,1], ncol=1 ),
                  matrix( res$sigma2.theta[,1], ncol=1 ),
                  matrix( res$sigma2.eps[,1], ncol=1 ) )
colnames( samples ) <- c( "mu", "sigma2.theta", "sigma2.eps" )
shinystan.obj <- as.shinystan( list( samples ) )

# retrieve effective sample size (ESS)
( ESS <- retrieve( shinystan.obj, "ess" ) )

# calculate MCMC efficiency (ESS/s)
( MCMC.efficiency <- ESS / as.numeric( runtime.sampling ) )
Appendix B: JAGS Code (Covariance- and Mean-based Parametrization)

# JAGS model code for the multi-level intercept-only model
# with the covariance- and mean-based parametrization

# Supplementary material to article:
# Hecht, M., Weirich, S., & Zitzmann, S. Comparing the MCMC efficiency of JAGS and Stan
# for the multi-level intercept-only model

model {

  # distributional assumption of sample scatter matrix (Equation 3)
  S ~ dwish( C, J-1 )

  ## construction of precision matrix C (Equation 7)
  # main diagonal
  for ( p in 1:P ){
    C[p,p] <- ((P-1)*sigma2.theta+sigma2.eps)/ksi
  }
  # lower/upper triangle
  for (k in 2:P) {
    for (l in 1:(k-1)) {
      C[k,l] <- -sigma2.theta/ksi
      C[l,k] <- C[k,l]
    }
  }
  # ksi (Equation 8)
  ksi <- P * sigma2.theta * sigma2.eps + sigma2.eps^2

  # precision matrix of means
  Cm <- J * C

  # distributional assumption of sample means (Equation 4)
  y.bar ~ dmnorm( mu.vec, Cm )

  # construction of mu vector
  for ( p in 1:P ){
    mu.vec[p] ~ mu
  }

  # prior distribution for mu
  mu ~ dnorm( 0, 1/10000 )
# prior distribution for $1/\sigma^2_{\text{theta}}$
prec.theta ~ dgamma( 0.001, 0.001 )
sigma2.theta ← 1/prec.theta

# prior distribution for $1/\sigma^2_{\text{eps}}$
prec.eps ~ dgamma( 0.001, 0.001 )
sigma2.eps ← 1/prec.eps

}
Appendix C: JAGS Code (Classic Parametrization)

# JAGS model code for the multi-level intercept-only model
# with the classic parametrization

# Supplementary material to article:
# Hecht, M., Weirich, S., & Zitzmann, S. Comparing the MCMC efficiency of JAGS and Stan
# for the multi-level intercept-only model

model {

  # loop over level 2 units
  for (j in 1:J) {
    # loop over level 1 units
    for (p in 1:P) {
      # distributional assumption of observations (Equation 1)
      D[p,j] ~ dnorm( theta[j], prec.eps )
    }

    # distributional assumption of the person parameters (Equation 2)
    theta[j] ~ dnorm( mu, prec.theta )
  }

  # prior distribution for mu
  mu ~ dnorm( 0, 1/10000 )

  # prior distribution for 1/sigma2.theta
  prec.theta ~ dgamma( 0.001, 0.001 )
  sigma2.theta <- 1/prec.theta

  # prior distribution for 1/sigma2.eps
  prec.eps ~ dgamma( 0.001, 0.001 )
  sigma2.eps <- 1/prec.eps
}
Appendix D: rstan Code

# R code for running the multi-level intercept-only model with Stan/rstan #

# Supplementary material to article:
# Hecht, M., Weirich, S., & Zitzmann, S. Comparing the MCMC efficiency of JAGS and Stan
# for the multi-level intercept-only model

## used R version: 4.0.5 (R Core Team, 2021)
# => get latest version: https://cran.r-project.org/

## load/install packages
# used rstan version: 2.21.2 (Stan Development Team, 2020)
# => get latest rstan version: install.packages("rstan")
library( rstan )

# used shinystan version: 2.5.0 (Gabry, 2018)
# => get latest shinystan version: install.packages("shinystan")
library( shinystan )

## working directory
# (!!! needs to include file 02b_multilevel_intonly_model_covmeanbased.stan !!!)
wd <- "SET_YOUR_WORKING_DIRECTORY_HERE___needs_to_include_file___02b_multilevel_intonly_model_covmeanbased.stan"

# for classic parametrization include: 02c_multilevel_intonly_model_classic.stan

## load simulated example data
# D: data matrix in wide format:
# rows: level 1 units (e.g., measurement occasions)
# columns: level 2 units (e.g., persons)
# cells: values
# J: number of level 2 units
# P: number of level 1 units

# load simulated example data
(load(url("https://amor.hu-berlin.de/~psymetho/jagsstan/00_multilevel_intonly_model_example_data.Rdata")))

# D: data matrix in wide format:
# rows: level 1 units (e.g., measurement occasions)
# columns: level 2 units (e.g., persons)
# cells: values
# J: number of level 2 units
# P: number of level 1 units

### preprocessing ###

## load simulated example data
# D: data matrix in wide format:
# rows: level 1 units (e.g., measurement occasions)
# columns: level 2 units (e.g., persons)
# cells: values
# J: number of level 2 units
# P: number of level 1 units

# load simulated example data
(load(url("https://amor.hu-berlin.de/~psymetho/jagsstan/00_multilevel_intonly_model_example_data.Rdata")))

# D: data matrix in wide format:
# rows: level 1 units (e.g., measurement occasions)
# columns: level 2 units (e.g., persons)
# cells: values
# J: number of level 2 units
# P: number of level 1 units

### preprocessing ###
## compute sample mean vector and sample scatter matrix from data

```r
# sample mean vector y.bar
y.bar <- matrix( rowMeans( D ), P, 1 )

# column vector of ones
ones.vec <- matrix( 1, J, 1 )

# sample scatter matrix
S <- ( D - y.bar %*% t(ones.vec) ) %*% t( ( D - y.bar %*% t(ones.vec) ) )
```

## data list for Stan

```r
# data list for Stan

data.list <- list()
data.list <- c( data.list, list( "S"=S ) )
data.list <- c( data.list, list( "ybar"=as.vector( y.bar ) ) )
data.list <- c( data.list, list( "J"=J ) )
data.list <- c( data.list, list( "P"=P ) )
data.list <- c( data.list, list( "D"=D ) ) # only needed for classic parametrization
```

---

```r
### model translation ###
```

```r
# translation start time
start.time.translation <- Sys.time()

# translate model
translated <- stanc( file = file.path( wd, "02b_multilevel_intonly_model_covmeanbased.stan" ), isystem = wd )

# for classic parametrization use: 02c_multilevel_intonly_model_classic.stan
# translated <- stanc( file = file.path( wd, "02c_multilevel_intonly_model_classic.stan" ), isystem = wd )

# translation runtime
(runtime.translation <- Sys.time() - start.time.translation)
```

---

```r
### model compilation ###
```

```r
# compilation start time
start.time.compilation <- Sys.time()

# compile model

translated <- stanc( file = file.path( wd, "02b_multilevel_intonly_model_covmeanbased.stan" ), isystem = wd )

# for classic parametrization use: 02c_multilevel_intonly_model_classic.stan
# translated <- stanc( file = file.path( wd, "02c_multilevel_intonly_model_classic.stan" ), isystem = wd )

# translation runtime
(runtime.compilation <- Sys.time() - start.time.compilation)
```
compiled <- stan_model( stanc_ret = translated, auto_write = TRUE, isystem = wd )

# compilation runtime
(runtime.compilation <- Sys.time() - start.time.compilation)

# compilation runtime

### warmup/sampling with function sampling() ###

# samplingfunction start time
(start.time.samplingfunction <- Sys.time())

# warmup/sampling iterations
(warmup.iterations <- 1000)
(sampling.iterations <- 100000)

# warmup/sampling
res <- sampling( object = compiled, data = data.list, chains = 1, cores = 1,
iter = warmup.iterations+sampling.iterations, warmup = warmup.iterations,
init = list( list( mu = 0, sigma2theta = 1, sigma2eps = 1 ) ),
seed = 12345 )

# optional for classic parametrization:
# add starting values (argument init) for parameters of level 2 units (theta)

# samplingfunction runtime
(runtime.samplingfunction <- Sys.time() - start.time.samplingfunction)

# times reported by rstan
runtimes <- attributes( res$sim$samples[[1]] )$"elapsed_time"

# time for warmup
(runtime.warmup <- runtimes["warmup"])

# time for sampling
(runtime.sampling <- runtimes["sample"])


# create shinystan object
samples <- cbind( matrix( res@sim[['samples']][[1]]$mu[(warmup.iterations+1):(warmup.iterations+sampling.iterations)], ncol=1 ),
                  matrix( res@sim[['samples']][[1]]$sigma2theta[(warmup.iterations+1):(warmup.iterations+sampling.iterations)], ncol=1 ),
                  matrix( res@sim[['samples']][[1]]$sigma2eps[(warmup.iterations+1):(warmup.iterations+sampling.iterations)], ncol=1 ) )
colnames( samples ) <- c( "mu", "sigma2.theta", "sigma2.eps" )
shinystan.obj <- as.shinystan( list( samples ) )

# retrieve effective sample size (ESS)
( ESS <- retrieve( shinystan.obj, "ess" ) )

# calculate MCMC efficiency (ESS/s)
( MCMC.efficiency <- ESS / as.numeric( runtime.sampling ) )
Appendix E: Stan Code (Covariance- and Mean-based Parametrization)

// Stan model code for the multi-level intercept-only model
// with the covariance- and mean-based parametrization

// Supplementary material to article:
// Hecht, M., Weirich, S., & Zitzmann, S. Comparing the MCMC efficiency
// of JAGS and Stan for the multi-level intercept-only model

data {
  matrix[20,20] S; // sample scatter matrix
  vector[20] ybar; // sample mean vector
  int<lower=1> J; // number of level 2 units
  int<lower=1> P; // number of level 1 units
}

parameters {
  real<lower=0> sigma2theta;
  real<lower=0> sigma2eps;
  real mu;
}

transformed parameters {
  matrix[20,20] Sigma;
  matrix[20,20] SigmaM;
  vector[20] muvec;

  // construction of covariance matrix Sigma (Equation 5)
  // main diagonal
  for ( p in 1:P ){
    Sigma[p,p] = sigma2theta + sigma2eps;
  }

  // lower/upper triangle
  for (k in 2:P){
    for (l in 1:(k-1)) {
      Sigma[k,l] = sigma2theta;
      Sigma[l,k] = Sigma[k,l];
    }
  }

  // covariance matrix of means
SigmaM = Sigma/J;

// construction of mu vector
for ( p in 1:P ){
    muvec[p] = mu;
}

model {

    // distributional assumption of sample scatter matrix (Equation 3)
    S ~ wishart( J-1, Sigma );

    // distributional assumption of sample means (Equation 4)
    ybar ~ multi_normal( muvec, SigmaM );

    // prior distribution for mu
    mu ~ normal( 0, sqrt(10000) );

    // prior distribution for sigma2theta
    sigma2theta ~ inv_gamma(0.001,0.001);

    // prior distribution for sigma2eps
    sigma2eps ~ inv_gamma(0.001,0.001);
}
Appendix F: Stan Code (Classic Parametrization)

```stan
// Stan model code for the multi-level intercept-only model
// with the classic parametrization

// Supplementary material to article:
// Hecht, M., Weirich, S., & Zitzmann, S. Comparing the MCMC efficiency
// of JAGS and Stan for the multi-level intercept-only model

data {
  int<lower=1> J; // number of level 2 units
  int<lower=1> P; // number of level 1 units
  matrix[P,J] D; // data matrix
}

parameters {
  real<lower=0> sigma2theta;
  real<lower=0> sigma2eps;
  real mu;
  vector[J] theta;
}

model {
  // loop over level 2 units
  for (j in 1:J) {
    // loop over level 1 units
    for (p in 1:P) {
      // distributional assumption of observations (Equation 1)
      D[p,j] ~ normal( theta[j], sigma2eps );
    }
    // distributional assumption of the person parameters (Equation 2)
    theta[j] ~ normal( mu, sigma2theta );
  }
  // prior distribution for mu
  mu ~ normal( 0, sqrt(10000) );

  // prior distribution for sigma2theta
  sigma2theta ~ inv_gamma(0.001,0.001);

  // prior distribution for sigma2eps
  sigma2eps ~ inv_gamma(0.001,0.001);
}
```
Appendix G: Simulation Code

# R code for running the simulation #

# Supplementary material to article:
# Hecht, M., Weirich, S., & Zitzmann, S. Comparing the MCMC efficiency of JAGS and Stan
# for the multi-level intercept-only model

## used R version: 4.0.5 (R Core Team, 2021)
# => get latest version: https://cran.r-project.org/

## used car version: 3.0-10
# => get latest car version: install.packages( "car" )
require(car)

## used rjags version: 4-10 (Plummer, 2019)
# => get latest rjags version: install.packages( "rjags" )
library(rjags)

## used rstan version: 2.21.2 (Stan Development Team, 2020)
# => get latest rstan version: install.packages( "rstan" )
library(rstan)

### function to generate data from the NULL model
## v1 21.05.2021

nullmodel_data_generation <- function( mu.true = 0, sigma2theta.true = 1, sigma2eps.true = 1, J = 1000, P = 20, seed=NULL ){

  # seed
  if( is.null( seed ) ) seed <- sample( 1:99999999, 1 )
  set.seed( seed )

  # data generation

thej <- rnorm( J, mu.true, sqrt(sigma2theta.true) )
y <- sapply( thetaj, function( mu ) rnorm( P, mu, sqrt(sigma2eps.true) ), simplify=TRUE )

# return
list( "y"=y, "seed"=seed )

### simulation conditions
ss <- 1000  ### sample size
nwarm<- c(150, 1000)  ### number of warmups
prior<- c(0.001, 0.01, 0.1, 1)  ### priors
iter <- 100000  ### number of iterations
reps <- 1:1000  ### number of replications
res <- expand.grid ( nWarmups = nwarm, prior = prior, iter = iter, replication = reps, sampleSize = ss,
stringsAsFactors = FALSE)
res <- data.frame ( res, sequential_number = 1:nrow(res), stringsAsFactors = FALSE)

### define paths, need approx. 45 GB free space on hard disc
dir <- "c:/stan_simulation"
outdir <- file.path(dir, "results")
workdir<- file.path(dir, "work")

### create folders if necessary
if(!dir.exists(outdir)) {dir.create(outdir)}
if(!dir.exists(workdir)) {dir.create(workdir)}

### jags and stan default model
jags <- scan(url("http://amor.hu-berlin.de/~psymetho/jagsstan/01b_multilevel_intonly_model.bugs"), what="character",sep="\n",quiet=TRUE)
stan <- scan(url("http://amor.hu-berlin.de/~psymetho/jagsstan/02b_multilevel_intonly_model.stan"), what="character",sep="\n",quiet=TRUE)

### R functions to start jags/stan functions from R
## v1 28.05.2021
# S ... sample scatter matrix
# y.bar ... sample mean column vector
# J ... number of level 2 units (persons)
# P ... number of level 1 units (observations/time points)
# n.iter ... number of iterations, default: 500
# warmup ... number of warmup iterations ("n.adapt"), default: 0
# bugs.file ... bugs/jags model syntax as plain text file, default: url("https://www.tandfonline.com/doi/suppl/10.1080/10705511.2019.1647432/suppl_file/hsem_a_1647432_sm6770.bugs")
# inits ... list (chains) of lists (parameters) with init values; "auto" (default) sets inits to mu=0, prec.
theta=1, prec.eps=1
# time.units ... character vector, time units, e.g. "secs" (default)
# value: list; erster Eintrag "res": data.frame (1. Spalte "parameter", weiter Spalten: Iterationen);
# zweiter Eintrag: named vector mit runtimes für warmup, run, total; dritter Eintrag "seeds": generierter
 Seed für den jags run
nullmodel_jags <- function( S, y.bar, J, P, n.iter=500, warmup=0, bugs.file, inits="auto", time.units="secs"
 ){
    ## data list for JAGS
    data.list <- list()
    data.list <- c( data.list, list( "S"=S ) )
    data.list <- c( data.list, list( "y.bar"=y.bar ) )
    data.list <- c( data.list, list( "J"=J ) )
    data.list <- c( data.list, list( "P"=P ) )
    ## JAGS run parameters
    # thinning
    thin <- 1
    # starting values
    if ( inits %in% "auto" ){
        # function to generate random starting values
        generate.startval <- function(){
            startval <- list()
            # startval <- c( startval, list( "mu" = rnorm(1,0,sqrt(10000)) ) )
            startval <- c( startval, list( "mu" = 0 ) )
            # startval <- c( startval, list( "prec.theta" = rgamma(1,0.001,0.001)
            +10^(-20) ) )
            startval <- c( startval, list( "prec.theta" = 1/1 ) )
            # startval <- c( startval, list( "prec.eps" = rgamma(1,0.001,0.001)+10^(-20)
            )
            startval <- c( startval, list( "prec.eps" = 1/1 ) )
            return( startval )
        }
        # generate starting values for the chain
        # set.seed( 1234 )
        inits <- sapply( 1:1, function (chain.nr,l) l, generate.startval(), simplify=
                        FALSE )
    }
    # set seeds for the chain
    seeds <- parallel.seeds( 'base::BaseRNG', 1 )
    inits <- mapply ( function(i,s) c(i,s), inits, seeds, SIMPLIFY=FALSE )
    ## initialize JAGS model
    # number of chains = 1 (code runs only for one chain)
# number of adaptation iterations = 0 (no pre-sampling, just initialization)
start.warmup <- Sys.time()
ini <- jags.model ( file=bugs.file, data=data.list, n.chains=1, n.adapt=warmup,
inits=inits, quiet=TRUE )

# run time warmup
runtime.warmup <- Sys.time() - start.warmup
units( runtime.warmup )<- time.units
runtime.warmup <- as.numeric( runtime.warmup )

# run JAGS model
track <- c("mu","sigma2.theta","sigma2.eps")
start.run <- Sys.time()
res <- jags.samples ( ini, variable.names=track, n.iter=n.iter, thin=thin,
type='trace', progress.bar = NULL )

# run time main run
runtime.run <- Sys.time() - start.run
units( runtime.run )<- time.units
runtime.run <- as.numeric( runtime.run )

# total run time
runtime <- runtime.warmup + runtime.run

# named vector with runtimes
runtimes <- c( runtime.warmup, runtime.run, runtime )

# results
dfr <- cbind( data.frame( "parameter"=names(res) ), do.call( "rbind", sapply( res, function(x)
x[1,,1], simplify=FALSE ) ) )
dfr <- dfr[ track, ]
rownames( dfr ) <- seq( along=rownames( dfr ) )

# return
list( "samples"=dfr, "runtimes"=runtimes, "seed"=seeds )

### start simulation
res <- by(data = res, INDICES = res[,"sequential_number"], FUN = function ( bed ) {
  dat <- nullmodel_data_generation(J=bed["sampleSize"],
  y.bar <- matrix( rowMeans( d ), 20, 1 ) ### mean vector (P = number of time points; D = matrix, persons x time points)
  ones <- matrix( 1, ncol(d), 1 ) ### vector with 1 values
  S <- ( d - y.bar %*% t(ones) ) %*% t( ( d - y.bar %*% t(ones) ) ) ### S matrix
  # first: jags. adapt priors in bugs file
  if(bed[,"prior"] != 0.001) {
    rows <- grep("dgamma", jags
  }
jags[rows] <- gsub("0.001", bed[,"prior"], jags[rows])
rows2<- grep("dnorm", jags)
jags[rows2] <- gsub("10000", car::recode(bed[,"prior"],"0.01 = 2500; 0.1 = 100; 1 = 25"), jags[rows2])

dir.create(file.path(workdir, paste0(bed[,"sequential_number"], ","jags")))
write(jags,file.path(workdir, paste0(bed[,"sequential_number"], ","jags"), "01b_multilevel_intonly_model.bugs"), sep="\n")

fit <- nullmodel_jags ( S=S, y.bar=y.bar, J=ncol(d), P=nrow(d), n.iter=bed[1,"iter"], warmup=bed[1,"nWarmups"], bugs.file = file.path(workdir, paste0(bed[,"sequential_number"], ","jags"), "01b_multilevel_intonly_model.bugs"), inits="auto", time.units="secs")

res.ja<- data.frame ( bed, software = "jags", runtime.translate = NA, runtime.compile = NA, runtime.samplingfunction = NA, t(fit["runtimes"]), stringsAsFactors = FALSE)

colnames(fit[["samples"]]) <- car::recode(paste0("c", colnames(fit[["samples"]])), "cparameter"='prm'

res.ja<- data.frame ( rbind(res.ja,res.ja,res.ja),fit["samples"], stringsAsFactors = FALSE)

# second: jags (use identical data). adapt priors in bugs file

if(bed[,"prior"] ! = 0.001) {
  rows <- grep("inv_gamma", stan)
  stan[rows] <- gsub("0.001", unique(bed[,"prior"]), stan[rows])
  rows2<- grep("normal\( 0", stan)
  stan[rows2] <- gsub("10000", unique(car::recode(bed[,"prior"],"0.01 = 2500; 0.1 = 100; 1 = 25")),
  stan[rows2])
}
dir.create(file.path(workdir, unique(paste0(bed[,"sequential_number"], ","stan"))))
write(stan,file.path(workdir, unique(paste0(bed[,"sequential_number"], ","stan")), "02b_multilevel_intonly_model.stan"), sep="\n")
dlist <- list(J = ncol(d), P = nrow(d), S=S, ybar = as.vector(y.bar)) ### stan does not allow points in object names

begin <- Sys.time() ### first step: translate
transl<- stanc(file = file.path(workdir, unique(paste0(bed[,"sequential_number"], ","stan"))), "02b_multilevel_intonly_model.stan"), isystem = file.path(workdir, unique(paste0(bed[,"sequential_number"], ","stan"))))
t1 <- Sys.time() - begin
units(t1) <- "secs"

begin2<- Sys.time() ### second step: compile
comp <- stan_model(stanc_ret = transl, auto_write = TRUE, isystem = file.path(workdir, unique(paste0(bed[,"sequential_number"], ","stan"))))
t2 <- Sys.time() - begin2
units(t2) <- "secs"

begin3<- Sys.time() ### third step: sampling
fit <- sampling(object = comp, data = dlist, iter = bed[1,"iter"] + bed[1,"nWarmups"], chains = 1,
```r
warmup = bed[1,"nWarmups"], init = list( list(mu=0, pretheta = 1, preeps = 1)), cores = 1)
t3  <- Sys.time() - begin3
units(t3) <- "secs"
tmes <- attributes(fit@sim$samples[[1]])['elapsed_time']
res.st <- data.frame ( bed, software="stan", runtime.translate = as.numeric(t1), runtime.compile = as.numeric(t2), runtime.samplingfunction = as.numeric(t3), runtime.warmup = tmes["warmup"], runtime.run = tmes["sample"], runtime = tmes["warmup"]+tmes["sample"], stringsAsFactors = FALSE)
chains<- as.data.frame(fit@sim["samples"][[1]])
chainSelect <- as.data.frame(t(chains[(bed[1,"nWarmups"]+1):nrow(chains),1:3]))
colnames(chainSelect) <- paste0("c", as.character(1:bed[1,"iter"]))
res.st[,"prm"] <- car::recode(rownames(chainSelect), "'sigma2theta'='sigma2.theta'; 'sigma2eps'='sigma2.eps'")
resAll<- rbind(res.ja, res.st)
save(resAll, file = file.path(outdir, paste0("results_",bed[,"sequential_number"])))
```
## Supplementary Material

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Currently available at:</th>
</tr>
</thead>
<tbody>
<tr>
<td>00_multilevel_intonly_model_example_data.Rdata</td>
<td>Simulated example data</td>
<td><a href="http://amor.hu-berlin.de/~psymetho/jagsstan/00_multilevel_intonly_model_example_data.Rdata">http://amor.hu-berlin.de/~psymetho/jagsstan/00_multilevel_intonly_model_example_data.Rdata</a></td>
</tr>
<tr>
<td>01a_rjags.R</td>
<td>rjags code to run JAGS models</td>
<td><a href="http://amor.hu-berlin.de/~psymetho/jagsstan/01a_rjags.R">http://amor.hu-berlin.de/~psymetho/jagsstan/01a_rjags.R</a></td>
</tr>
<tr>
<td>01b_multilevel_intonly_model_covmeanbased.bugs</td>
<td>JAGS model code (covariance- and mean-based parametrization)</td>
<td><a href="http://amor.hu-berlin.de/~psymetho/jagsstan/01b_multilevel_intonly_model_covmeanbased.bugs">http://amor.hu-berlin.de/~psymetho/jagsstan/01b_multilevel_intonly_model_covmeanbased.bugs</a></td>
</tr>
<tr>
<td>01c_multilevel_intonly_model_classic.bugs</td>
<td>JAGS model code (classic parametrization)</td>
<td><a href="http://amor.hu-berlin.de/~psymetho/jagsstan/01c_multilevel_intonly_model_classic.bugs">http://amor.hu-berlin.de/~psymetho/jagsstan/01c_multilevel_intonly_model_classic.bugs</a></td>
</tr>
<tr>
<td>02a_rstan.R</td>
<td>rstan code to run Stan models</td>
<td><a href="http://amor.hu-berlin.de/~psymetho/jagsstan/02a_rstan.R">http://amor.hu-berlin.de/~psymetho/jagsstan/02a_rstan.R</a></td>
</tr>
<tr>
<td>02b_multilevel_intonly_model_covmeanbased.stan</td>
<td>Stan model code (covariance- and mean-based parametrization)</td>
<td><a href="http://amor.hu-berlin.de/~psymetho/jagsstan/02b_multilevel_intonly_model_covmeanbased.stan">http://amor.hu-berlin.de/~psymetho/jagsstan/02b_multilevel_intonly_model_covmeanbased.stan</a></td>
</tr>
<tr>
<td>02c_multilevel_intonly_model_classic.stan</td>
<td>Stan model code (classic parametrization)</td>
<td><a href="http://amor.hu-berlin.de/~psymetho/jagsstan/02c_multilevel_intonly_model_classic.stan">http://amor.hu-berlin.de/~psymetho/jagsstan/02c_multilevel_intonly_model_classic.stan</a></td>
</tr>
<tr>
<td>03_simulation.R</td>
<td>R simulation code</td>
<td><a href="http://amor.hu-berlin.de/~psymetho/jagsstan/03_simulation.R">http://amor.hu-berlin.de/~psymetho/jagsstan/03_simulation.R</a></td>
</tr>
</tbody>
</table>

1 All supplementary files will be uploaded to the journal’s website after publication.
Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

Author Contributions

The authors declare the following contributions (as defined by http://credit.niso.org) to this article:

Author 1: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Software, Supervision, Validation, Visualization, Writing: original draft, Writing: review & editing,

Author 2: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing: review & editing,

Author 3: Conceptualization, Methodology, Supervision, Validation, Writing: review & editing.

Conflict of interest

The authors declare no conflict of interest.

Ethics statement

In this study, no human participants nor animals were involved.