COMPARISON OF SOFTWARE PACKAGES FOR PERFORMING BAYESIAN INFERENCE

*M. Koprivica**

Abstract: In this paper, we compare three state-of-the-art Python packages for Bayesian inference: JAGS [14], Stan [5], and PyMC3 [18]. These packages are in focus because they are the most mature, and Python is among the most utilized programming languages for teaching mathematics and statistics in colleges [13]. The experiment is based on real-world data collected for investigating the therapeutic touch nursing technique [17]. It is analyzed through a hierarchical model with prior beta distribution and binomial likelihood function. The tools are compared by execution time and sample quality.

Key words: *Bayesian statistics, hierarchical model, parameter estimation, Markov chain Monte Carlo, JAGS, Stan, PyMC3*

Received: June 19, 2020 **DOI:** 10.14311/NNW.2020.30.019
Revised and accepted: October 30, 2020

1. Introduction

The essence of Bayesian inference as a statistical inference method is using Bayes' rule to update previous knowledge, usually referred to as prior, with new data. However, as the calculation of the evidence integral in Bayes' rule can be knotty, for solving Bayesian inference problems in real-world applications, it is convenient to use MCMC (Markov chain Monte Carlo) simulation algorithm [2]. This study evaluates the performance of three Bayesian inference programs, namely JAGS, Stan, and PyMC3. They are currently state-of-the-art in the field. We aim to rank packages by execution duration and the quality of samples.

For the duration, the experiment measures the time needed for sampling 20,000 samples and a 1,000-sample warmup. As for quality, we are interested in finding the independence of information in the posterior distribution, measured by the effective sample size [8] and Monte Carlo standard error [4], in accessing how stable the sample mean is. As MCMC is a stochastic process, we collect data from 100 repetitions and draw conclusions from statistics.

In his research, Almond [1] compared JAGS and Stan, called from R programming language, for sampling from a hierarchical mixture model. As opposed to that paper, our focus is on Python programming language, and we are extending comparison by measuring PyMC3 performance. Also, we are interested in evaluating the stochastic process managed by software packages.

*Marko Koprivica; Faculty of Organizational Sciences, Department of Software Engineering, Belgrade University, Jove Ilica 154, Belgrade, Serbia, E-mail: marko.koprivica@hotmail.com

The rest of the paper is organized as follows: In Section 2, we present software packages that are compared; Section 3 presents an experiment scenario which is used for comparison; Section 4 presents experiment results, and Section 5 the conclusions.

2. Software packages

In this segment we give a summary of the software packages being compared. All of them are accessible as Python packages that can be easily imported and used. Only JAGS stands out as independent software that needs to be separately installed on the computer and for which Python package is just an interface. JAGS [14] is an abbreviation from "Just another Gibbs sampler". It is a program written in C++, to analyze Bayesian models using Markov chain Monte Carlo (MCMC). It is designed to work closely with the R language and environment for statistical computation and graphics [15]. We are using the Python library "pyjags" version 1.2.2. Stan [5], named after Stanislaw Ulam, is an open-source C++ program that performs Bayesian inference or optimization for arbitrary user-specified models and can be called from the command line, R, Python, MATLAB, or Julia and has great promise for fitting large and complex statistical models in many areas of application [5]. Compared to JAGS, Stan is more flexible because its modelling language is more general [5]. Our tests are performed with the Python library "pystan" version 2.18.1.0. PyMC3 [18] is an open-source probabilistic programming framework written in Python that uses Theano [19] to compute gradients via automatic differentiation and compile probabilistic programs on-the-fly to C programming language for increased speed. We are using the Python library "pymc3" version 3.7.

Both PyMC3 and JAGS can use several sampling methods (such as Gibbs [6], Metropolis [10], Slice sampling [11]), and JAGS can choose which one to use depending on the task [3]. For PyMC3, as advised [18], we are using Hamiltonian Monte Carlo (HMC) with No-U-Turn Sampler (NUTS) [7]. On the other hand, Stan's Markov chain Monte Carlo (MCMC) techniques are solely based on HMC with NUTS [5, 7].

3. Experiment

Therapeutic touch (TT) is a widely used nursing practice rooted in mysticism but alleged to have a scientific basis. Practitioners of TT claim to treat many medical conditions by using their hands to manipulate a "human energy field" perceptible above the patient's skin [17]. Rosa investigated this claim by conducting research in which the practitioner was hovering with his hand over the left or right hand of the experimenter, and the experimenter should guess which hand is it. Results were collected for 28 practitioners with ten trails for each of them, making 280 records in total [17]. These records are new data in our Bayesian inference experiment.

3.1 Model

Let $s \in 1, \ldots, 28$ be a set of indexes over practitioners, then for each practitioner: N_s is the number of trials, z_s is the number of good guesses, and θ_s is a parameter of distribution that describes the underlying ability. Since the data are binary (0 for wrong and 1 for correct), chance performance is 0.5. As advised by Kruschke [9], to distinguish how much the group as a whole and any practitioner differed from chance performance, we are implementing the hierarchical model with binomial distribution used as likelihood:

$$z_s \sim Binom(\theta_s, N_s),$$

and beta distribution applied to the hierarchical model's first level:

$$\theta_s \sim Beta(\omega(\kappa - 2) + 1, (1 - \omega)(\kappa - 2) + 1).$$

The latest is used because Kruschke [9] found that the distribution of proportions across subjects is essentially unimodal and can be meaningfully described as a beta distribution. On the second level, we estimate the group's modal ability and the consistency of the group with beta distribution for mode:

$$\omega \sim Beta(1, 1),$$

and gamma distribution for concentration:

$$\kappa + 2 \sim Gamma(0.01, 0.01).$$

We estimate 30 parameters, 28 thetas – one for each practitioner, and omega and kappa. The model is simple enough to demonstrate what can be expected from software packages in everyday work, and yet it is not trivial, so differences between software packages are measurable. The model is depicted as a directed acyclic graph (DAG) below, in Fig. 1.

3.2 JAGS

JAGS' model is straightforward; first, we describe theta and binomial likelihood for each practitioner. Next goes second layer distributions; omega, as a result of sampling from a beta distribution with shape parameters both equal to 1, and kappa, as a result of sampling from a gamma distribution with shape parameter equals to 0.01 and rate parameter equals to 0.01.

```
model {
  for (s in 1:Nsubj) {
    z[s] ~ dbin(theta[s], N[s])
    theta[s] ~ dbeta(omega*(kappa-2)+1, (1-omega)*(kappa-2)
    ↪ +1)
  }
  omega ~ dbeta(1,1)
  kappa <- kappaMinusTwo + 2
  kappaMinusTwo ~ dgamma(0.01 , 0.01)
}
```

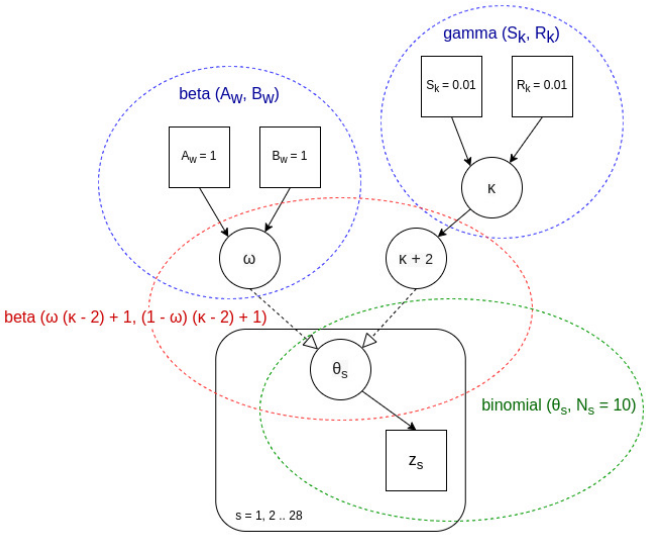

Fig. 1 *DAG contains a hierarchical model with beta distribution on the first level of the model. The second level is beta and gamma distributions.*

Because kappa must be greater than two, we do the trick, add two to the sample from a gamma distribution.

3.3 Stan

With Stan, we have a bit different situation. First, we have to define data variables: the number of practitioners ($Nsubj$), and for each practitioner, the number of tries (N) and the number of good guesses (z). Next, we define parameters that are sampled from distributions: theta for each practitioner, omega, and kappa. Following is a description of the transformed parameter because we are adding two on the kappa sample so we can be sure that it is greater than two. Finally, we come to the model in which distributions and likelihood are defined.

```
data {
  int<lower=1> Nsubj;
  int N[Nsubj];
  int z[Nsubj];
}
parameters {
  real<lower=0, upper=1> theta[Nsubj];
  real<lower=0, upper=1> omega;
  real<lower=0> kappaMinusTwo;
}
```

```
transformed parameters {
  real<lower=0> kappa;
  kappa = kappaMinusTwo + 2;
}
model {
  omega ~ beta(1,1);
  kappaMinusTwo ~ gamma(0.01 , 0.01);
  theta ~ beta(omega*(kappa-2)+1 , (1-omega)*(kappa-2)+1);
  z ~ binomial(N, theta);
}
```

3.4 PyMC3

Unlike from the previous two software packages, PyMC3 does not have a model, or data written as a string that is then passed to the actual executor (program installed for JAGS or, in the case of Stan, C++ library). With PyMC3, everything is coded directly in Python. First, we define higher-level distributions and parameters: omega and kappa. Next, we do the trick by adding two to kappa. Finally, we define theta and likelihood. The number of good guesses is stored in the Nz variable.

```
with pm.Model() as model:
  omega = pm.Beta('omega', 1, 1, testval=meanThetaInit)
  kappaMinusTwo = pm.Gamma('kappa', 0.01, 0.01, testval=
    ↪ kappaInitMinusTwo)

  kappa = kappaMinusTwo + 2

  theta = pm.Beta('theta', omega * (kappa - 2) + 1, (1 -
    ↪ omega) * (kappa - 2) + 1, shape=self.Nsubj, testval=
    ↪ np.array(ThetaI))

  self.z = pm.Binomial('z', p=theta, n=np.array(N), observed
    ↪ =np.array(Nz))
```

3.5 Settings

We set three chains for each software package, with 20 thinning and 1,000 burn-in steps. We are saving 20,000 steps for PyMC3 and 20,001 steps for JAGS and Stan. The difference in the numbers of saved steps is due to lack of thinning implementation in PyMC3, so extraction of every twentieth step is done after steps from all three chains are collected in one place, whereas JAGS and Stan are doing thinning on each chain and then merge resulting steps. Initial parameter values for all three packages are the same. For theta and omega, maximum likelihood estimation (MLE) is calculated, while kappa is fixed at 98. To calculate MLE for theta, we randomly pick ten numbers (number of trials for each practitioner) from data that we have for that practitioner. Next, we sum the numbers and divide by 10 to get the average. We do a simple trick to keep the theta value inside the

valid range, away from 0.1 and 1, by multiplying it with 0.998 and after adding 0.001. This trick is preventing the situations in which the chain cannot be started due to zero density of prior beta distribution [9]. Omega is calculated as the mean of theta for all practitioners, and for that calculation, we are using NumPy [12] Python library.

```
import numpy as np

for practitioner in range(0, 28):

  initList[practitioner] = np.random.choice((therapeutic-
    ↪ thouch-data[practitioner]).values, size=10, replace=
    ↪ True, pNone)

  ThetaInitial[practitioner] = sum(initList[practitioner])
    ↪  / len(initList[practitioner])

  ThetaInitial[practitioner] = 0.001 + 0.998 * ThetaI[
    ↪ practitioner]

  practitioner += 1

omegaInitial= np.mean(ThetaI)
```

We collected execution duration and parameter estimations of 100 runs for each software package. The computer used for the experiment has 8GB of RAM and an Intel i7 processor with eight threads.

4. Test results

4.1 Duration

Tab. I reports the mean, median, and standard deviation of execution duration over 100 probes for all three software packages. JAGS has the best results overall with a mean time of 98.66 seconds. Its median is close to the mean, and the standard deviation is tiny, so we can say that the mean is a good representation of the time necessary for JAGS to finish sampling.

Stan is taking second place with a mean time of 182.45 seconds, but its median is a bit far from the mean, and the standard deviation is considerable compared to the other values. By inspecting data, we can see one outlier which pulled mean and standard deviation. Without it, the standard deviation would be just 7.86 seconds.

PyMC3 is much slower than the other two packages, and its standard deviation shows that results are quite unstable.

4.2 Sample quality

After collecting samples for each software package, we are using the Coda package [16] to calculate the effective sample size (ESS) [8], where the higher value is better,

	JAGS	*Stan*	PyMC3
Mean	98.66	182.45	1046.44
Median	98.50	157.74	908.05
SD	1.60	238.44	431.10

Tab. I *Mean, median, and standard deviation for execution duration over 100 probes. Values are in seconds.*

and Monte Carlo standard error (MCSE) [4], where the lower value is better, for each parameter. Further, we are calculating mean, median, and standard deviation over the probes.

Fig. 2 is a chart with statistics of an effective sample size for theta parameters. The solid line presents the value of the mean, and the dotted line presents the standard deviation. Stan's values are both, encircling the other two packages' results in case of mean and are encircled by the other two packages in case of standard deviation. Also, we can see that the values of PyMC3 standard deviation significantly stands out. Fig. 3 is a chart with statistics of Monte Carlo standard error, where again mean is presented with a solid line while the standard deviation is presented with a dotted line. Here, PyMC3 values of both mean and standard deviation are encircling values of the other two packages. Finally, in Tab. II are presented the results of the last two parameters: omega and kappa. They are following the results of theta parameters; Stan has the highest mean and the lowest

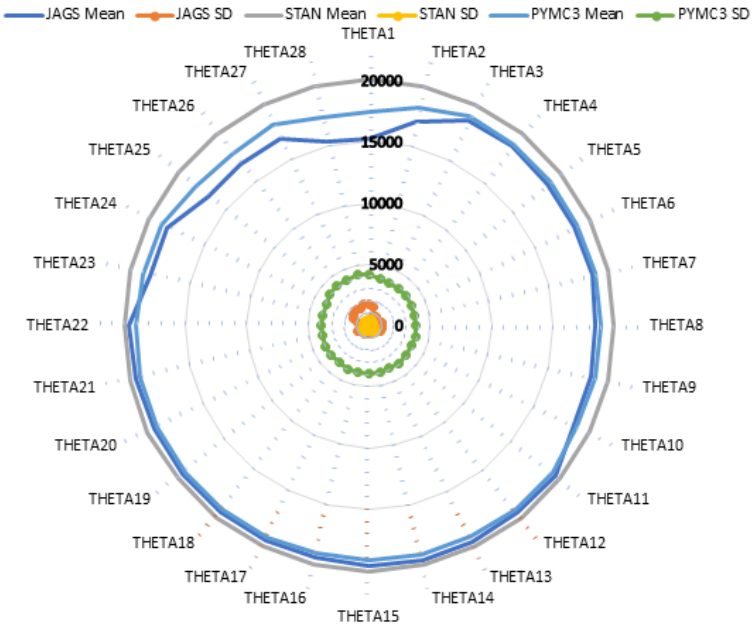

Fig. 2 *Mean and standard deviation of effective sample size across theta parameters for JAGS, Stan, and PyMC3.*

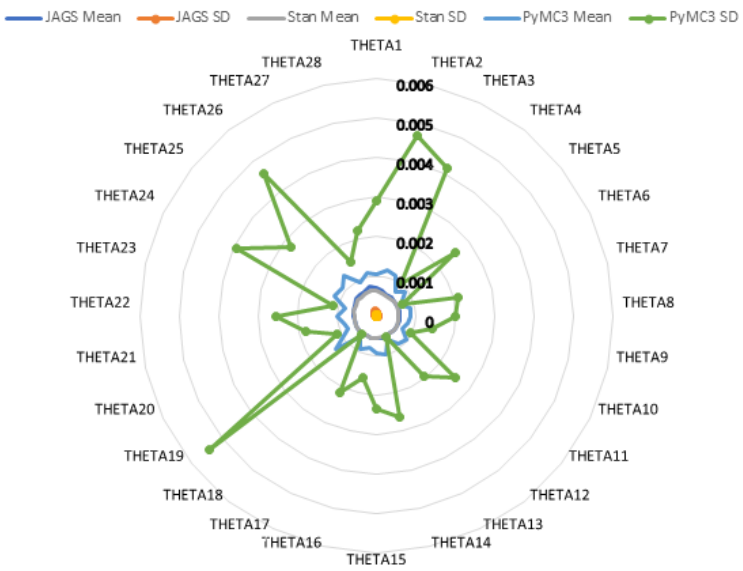

Fig. 3 *Mean and standard deviation of Monte Carlo standard error across theta parameters for JAGS, Stan, and PyMC3.*

standard deviation of effective sample size values, and PyMC3 has the highest values of the mean and standard deviation of Monte Carlo standard error values. All collected ESS and MCSE statistics for theta, kappa and omega parameters are presented in Tab. III and Tab. IV in the appendix.

Without question, Stan has the best results. The median of its ESS is equal to the total number of samples. The mean is sometimes greater than the total number of samples, indicating a negative estimated autocorrelation. The standard deviation is low, about 0.025 % of the total number of samples, which signifies that ESS is clustered around the mean. MCSE is following ESS; its mean and standard deviation are the lowest among packages. Even though JAGS mean and median are lower for about two-thirds of parameters from PyMC3 and higher for others, JAGS standard deviation is much lower than from PyMC3, which means that over the probes, ESS of JAGS samples is closer to the mean than in the case of PyMC3. MCSE values are entirely in favour of JAGS over PyMC3. PyMC3 has the highest MCSE numbers, with the highest standard deviation among packages.

5. Conclusion

This paper discussed the comparison of software packages for performing Bayesian inference in the Python programming language. We presented sampling results from the hierarchical model with the use of real data from the Therapeutic Touch experiment. We conclude that JAGS is the fastest among packages, with fairly good sampling quality. The standard deviation of both duration and ESS or MCSE makes JAGS very reliable. The best quality results produced Stan, but it requires

		OMEGA			KAPPA		
		JAGS	Stan	PyMC3	JAGS	Stan	PyMC3
ESS	Mean	16541.8	20053.1	18430.2	6927.1	19998.4	10928.0
	SD	2171.8	418.0	4202.1	984.5	581.3	4091.2
MCSE	Mean	0.000299	0.000269	0.000397	0.681980	0.400802	0.735324
	SD	0.000064	0.000011	0.000713	0.064937	0.010338	1.088703

Tab. II *Mean and standard deviation of effective sample size and Monte Carlo standard error of omega and kappa parameters, for JAGS, Stan, and PyMC3.*

more time for sampling than JAGS. If quality is the only requirement, then Stan is the best choice. These findings are aligned with Russel's for JAGS and Stan, using the R programming language [1]. PyMC3 acted poorly in this experiment. Even though it uses the same MCMC algorithm as Stan, as both are using HMC with NUTS, the standard deviation of its samples is significantly larger than for the other two competitors. Its only advantage is writing the model directly in Python and not as string-like for JAGS and Stan. This feature makes it easier to find modelling errors.

We plan to run more experiments with the objective of getting more parameter estimates. Also, the plan is to experiment with scenarios for which prior and likelihood must be expressed with custom functions.

Acknowledgement

The author would like to thank professor D. Djurić for the comments and discussion provided while preparing this work.

Program availability

The source code of the program written in Python for this experiment and data results is available from https://github.com/koprivica/MCMC-modules-comparison under the MIT license.

References

[1] ALMOND R. A Comparison Of Two Mcmc Algorithms For Hierarchical Mixture Models. In: *Proceedings of the 11th Uncertainty in Artificial Intelligence Conference on Bayesian Modeling Applications Workshop*, Quebec City, Quebec, Canada. 2014, pp. 1–19. Retrieved from: http://ceur-ws.org/Vol-1218/.

[2] BESAG J., GREEN P., HIGDON D., MENGERSEN K. Bayesian Computation and Stochastic Systems. *Statistical Science*. 1995, 10(1), pp. 3–66.

[3] DEPAOLI S., CLIFTON J.P., COBB P.R. Just Another Gibbs Sampler (JAGS): Flexible software for MCMC implementation. *Journal of Educational and Behavioral Statistics*. 2016, 41(6), pp. 628–649, doi: 10.3102/1076998616664876.

[4] FLEGAL J.M., HARAN M., JONES G.L. Markov chain Monte Carlo: Can we trust the third significant figure?. *Statistical Science*. 2008, 23(2), pp. 250–260, doi: 10.1214/08-STS257.

[5] GELMAN A., LEE D., GUO J. Stan: A probabilistic programming language for Bayesian inference and optimization. *Journal of Educational and Behavioral Statistics*. 2015, 40(5), pp. 530–543, doi: 10.3102/1076998615606113.

[6] GEMAN S., GEMAN D. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1984, 6, pp. 721–741.

[7] HOFFMAN M.D., GELMAN A. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *The Journal of Machine Learning Research*. 2014, 15(1), pp. 1593–1623.

[8] KASS R.E., CARLIN B.P., GELMAN A., NEAL R.M. Markov chain Monte Carlo in practice: A roundtable discussion. *The American Statistician*. 1998, 52, pp. 93–100, doi: 10.1080/00031305.1998.10480547.

[9] KRUSCHKE J.K. *Doing Bayesian Data Analysis (Second Edition)*. New York, NY: Academic Press. 2015.

[10] METROPOLIS, N., ROSENBLUTH A.W., ROSENBLUTH M.N., TELLER A.H., TELLER E. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*. 1953, 21, pp. 1087–1091.

[11] NEAL R.M. Slice Sampling. *The Annals of Statistics*. 2003, 31(3), pp. 705–767, doi: 10.1214/aos/1056562461.

[12] OLIPHANT T.E. *Guide to NumPy*. Trelgol Publishing USA. 2006.

[13] OZGUR C., COLLIAU T., ROGERS G., HUGHES Z., MYER-TYSON B. Matlab vs. python vs. r. *Journal of Data Science*. 2017, 15(3), pp. 355–372, doi: 10.13140/RG.2.1.2933.8480.

[14] PLUMMER M. 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)*, Vienna, Austria; 2003, Retrieved from: http://www.ci.tuwien.ac.at/Conferences/DSC-2003/.

[15] PLUMMER M. JAGS Version 4.3.0 User Manual. 2017, Retrieved from: https://people.stat.sc.edu/hansont/stat740/jags_user_manual.pdf.

[16] PLUMMER M., BEST N., COWLES K., VINES K. CODA: Convergence diagnosis and output analysis for MCMC *R News*. 2006, 6(1), pp. 7–11.

[17] ROSA L., ROSA E., SARNER L., BARRETT S. A close look at therapeutic touch. *Journal of the American Medical Association*. 1998, 279(13), pp. 1005–1010, doi: 10.1001/jama.279.13.1005.

[18] SALVATIER J., WIECKI T.V., FONNESBECK C. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*. 2016, 2, e55, doi: 10.7717/peerj-cs.55.

[19] The Theano Development Team: Theano: A python framework for fast computation of mathematical expressions. 2016, arXiv preprint, arXiv: 1605.02688.

Appendix

	JAGS			Stan			PyMC3		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
KAPPA	6927.1	6935.1	984.5	19998.4	20001.0	581.3	10928.0	12057.3	4091.2
OMEGA	16541.8	17060.1	2171.8	20053.1	20001.0	418.0	18430.2	19816.6	4202.1
THETA1	15208.1	15475.0	1769.3	20058.7	20001.0	346.8	17465.0	18809.5	4177.3
THETA2	17080.4	17270.7	1513.6	19996.5	20001.0	434.7	18259.6	19383.3	3915.0
THETA3	18588.1	18661.9	910.6	20045.6	20001.0	488.1	18989.3	20000.0	3828.4
THETA4	18768.7	18786.8	840.9	20092.5	20001.0	392.2	18867.0	20000.0	3865.1
THETA5	18576.2	18554.4	687.5	20016.8	20001.0	284.4	18946.2	20000.0	3852.3
THETA6	18547.2	18658.6	921.4	20032.5	20001.0	415.5	18858.9	20000.0	3853.1
THETA7	18699.3	18865.8	1026.3	20071.6	20001.0	391.2	19013.7	20000.0	3803.0
THETA8	18541.7	18605.9	988.7	20011.5	20001.0	388.0	18922.4	20000.0	3849.2
THETA9	18635.5	18736.4	1067.3	20042.8	20001.0	513.8	19023.8	20000.0	3884.5
THETA10	18605.0	18754.4	994.7	19975.8	20001.0	538.9	18904.2	20000.0	3827.9
THETA11	19548.9	20001.0	663.1	19993.3	20001.0	365.9	19192.8	20000.0	3901.7
THETA12	19517.1	19798.1	664.6	20030.5	20001.0	703.9	19213.2	20000.0	3928.1
THETA13	19562.1	19655.6	645.6	19969.4	20001.0	278.9	19167.1	20000.0	3815.1
THETA14	19676.4	20001.0	726.2	20017.3	20001.0	411.0	19185.5	20000.0	3933.3
THETA15	19658.6	20001.0	653.3	20081.7	20001.0	399.1	19139.5	20000.0	3898.0
THETA16	19478.8	19739.9	685.3	20066.7	20001.0	409.9	19097.8	20000.0	3893.7
THETA17	19542.6	19780.9	706.2	20012.6	20001.0	425.1	19187.1	20000.0	3927.4
THETA18	19479.6	19708.5	736.8	20039.5	20001.0	590.7	19218.4	20000.0	3871.9
THETA19	19535.6	19702.5	611.4	19977.3	20001.0	551.2	19184.7	20000.0	3930.8
THETA20	19480.3	20001.0	966.0	20157.2	20001.0	558.3	19173.2	20000.0	3909.4
THETA21	19555.1	20001.0	699.6	20013.3	20001.0	453.8	19224.1	20000.0	3857.1
THETA22	19638.2	20001.0	616.0	20019.8	20001.0	399.9	19112.7	20000.0	3937.0
THETA23	18368.1	18510.0	944.2	20072.9	20001.0	476.7	18936.1	20000.0	3914.6
THETA24	18293.6	18464.0	1374.9	20018.8	20001.0	407.0	18870.0	20000.0	3881.9
THETA25	16825.5	17161.4	1562.0	19993.4	20001.0	273.1	18137.2	19417.9	4173.3
THETA26	16920.2	17179.9	1690.0	19987.7	20001.0	520.1	17961.0	19232.0	4205.7
THETA27	16871.2	16931.1	1568.8	20074.3	20001.0	354.6	18161.1	19358.2	4162.9
THETA28	15407.5	15676.1	1819.1	20062.1	20001.0	471.2	17441.6	18912.1	4321.2

Tab. III *Statistics for effective sample size of JAGS, Stan, and PyMC3 over 100 probes.*

	JAGS			Stan			PyMC3		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
KAPPA	0.681980	0.673900	0.064937	0.400802	0.399788	0.010338	0.735324	0.510221	1.088703
OMEGA	0.000299	0.000286	0.000064	0.000269	0.000266	0.000011	0.000397	0.000270	0.000713
THETA1	0.000723	0.000707	0.000097	0.000622	0.000623	0.000007	0.001054	0.000642	0.002927
THETA2	0.000641	0.000632	0.000048	0.000589	0.000589	0.000007	0.001186	0.000598	0.004672
THETA3	0.000590	0.000588	0.000017	0.000567	0.000567	0.000007	0.001126	0.000570	0.004163
THETA4	0.000587	0.000586	0.000016	0.000567	0.000568	0.000006	0.000769	0.000572	0.001023
THETA5	0.000590	0.000588	0.000012	0.000567	0.000567	0.000005	0.000945	0.000571	0.002578
THETA6	0.000590	0.000588	0.000018	0.000568	0.000568	0.000007	0.000707	0.000571	0.000713
THETA7	0.000588	0.000585	0.000021	0.000567	0.000568	0.000006	0.000882	0.000571	0.002116
THETA8	0.000590	0.000587	0.000019	0.000568	0.000567	0.000006	0.000872	0.000571	0.002013
THETA9	0.000589	0.000586	0.000020	0.000568	0.000568	0.000008	0.000804	0.000570	0.001445
THETA10	0.000589	0.000586	0.000020	0.000569	0.000568	0.000008	0.000742	0.000571	0.000976
THETA11	0.000566	0.000563	0.000010	0.000560	0.000560	0.000006	0.000988	0.000560	0.002540
THETA12	0.000567	0.000565	0.000010	0.000559	0.000560	0.000010	0.000883	0.000560	0.001959
THETA13	0.000566	0.000565	0.000009	0.000561	0.000560	0.000005	0.000667	0.000561	0.000574
THETA14	0.000564	0.000562	0.000012	0.000559	0.000559	0.000007	0.001024	0.000562	0.002618
THETA15	0.000564	0.000562	0.000010	0.000559	0.000559	0.000006	0.000960	0.000560	0.002370
THETA16	0.000573	0.000570	0.000011	0.000564	0.000565	0.000006	0.000831	0.000566	0.001607
THETA17	0.000572	0.000570	0.000012	0.000566	0.000565	0.000007	0.000935	0.000566	0.002147
THETA18	0.000573	0.000571	0.000012	0.000565	0.000565	0.000009	0.000685	0.000567	0.000610
THETA19	0.000573	0.000571	0.000010	0.000566	0.000566	0.000008	0.001327	0.000566	0.005438
THETA20	0.000574	0.000568	0.000018	0.000563	0.000565	0.000008	0.000784	0.000566	0.001108
THETA21	0.000572	0.000570	0.000012	0.000565	0.000565	0.000008	0.000845	0.000566	0.001831
THETA22	0.000571	0.000568	0.000010	0.000566	0.000566	0.000006	0.000999	0.000568	0.002542
THETA23	0.000611	0.000607	0.000019	0.000584	0.000584	0.000008	0.000806	0.000588	0.001126
THETA24	0.000614	0.000607	0.000038	0.000585	0.000585	0.000007	0.001193	0.000589	0.003948
THETA25	0.000676	0.000664	0.000063	0.000617	0.000616	0.000006	0.001122	0.000627	0.002784
THETA26	0.000675	0.000666	0.000076	0.000617	0.000617	0.000009	0.001322	0.000628	0.004602
THETA27	0.000676	0.000669	0.000069	0.000616	0.000616	0.000006	0.000923	0.000626	0.001503
THETA28	0.000761	0.000743	0.000120	0.000658	0.000658	0.000010	0.001100	0.000677	0.002206

Tab. IV *Statistics for Monte Carlo standard error for JAGS, Stan, and PyMC3 over 100 probes.*