Introducing the StataStan interface for fast, complex Bayesian modeling using Stan

Robert L Grant[1], Bob Carpenter[2], Daniel Furr[3], and Andrew Gelman[2]

[1]*Faculty of Health, Social Care and Education, Kingston University and St George's, University of London, Cranmer Terrace, London SW17 0RE, United Kingdom*
[2]*Columbia University*
[3]*University of California at Berkeley*

December 5, 2016

Abstract

We present StataStan, an interface that allows simulation-based Bayesian inference in Stata via calls to Stan, the flexible open-source Bayesian inference engine. Stan is written in C++, and Stata users can use the commands `stan` and `windowsmonitor` to run Stan programs from within Stata We provide a brief overview of Bayesian algorithms, details of the commands, which are available from SSC, considerations for users who are new to Stan, and a simple example. Stan uses a different algorithm to `bayesmh`, BUGS, JAGS, SAS and MLwiN, which provides considerable improvements in efficiency and speed. A companion paper gives an extended comparison of StataStan and `bayesmh` in the context of item-response theory models.

Keywords: Bayesian; `bayesmh`; interface; shell commands; Stan.

1 Introduction

Stata users have long been able to seamlessly access other software specialising in Bayesian analysis, thanks to Stata's provision for the user to write arbitrary information to ASCII text files, and to send commands to the operating system. This allowed for programs such as `runmlwin` and `wb` to send data and code to MLwiN and WinBUGS respectively, and then collect the results and display them inside Stata and make them available for further calculation and graphing [1,2]. Since version 14 was released in 2015, Stata users have been able to utilise a native implementation of Bayesian simulation algorithms via the `bayesmh` command [3]. However, `bayesmh` is focussed on regression models and, outside that framework, users must define their own likelihood evaluators.

The day after version 14's release, StataStan was published online [4]. StataStan is an umbrella term for all commands and programs necessary to interface with Stan from Stata. Stan is an open-source, collaboratively-built software project to implement a newer, faster and more stable algorithm (Hamiltonian Monte Carlo) for Bayesian modelling than the algorithms (random walk Metropolis-Hastings and the Gibbs Sampler) implemented in BUGS, JAGS, SAS, MLwiN and `bayesmh`. Stan has been applied to a wide range of complex statistical models including time series, imputation, mixture models, meta-analysis, cluster analysis, Gaussian processes and item-response theory. These extend beyond the current (Stata 14.1) capability of `bayesmh`, which is explicitly for regression [3]. The functionality of Stan and advantages of the algorithm are described in our companion paper [5]. This paper gives a brief overview of Hamiltonian Monte Carlo in intuitive terms, then sets out the syntax of the commands, and finally presents a worked example.

2 Hamiltonian Monte Carlo

All Bayesian methods make estimates and inference by evaluating posterior distributions, combinations of likelihood based on data and a model with prior distributions representing uncertainty about parameters of the model before the data were known [3]. Different practitioners take the prior to mean different concepts, in the same way that 'uncertainty' or 'probability' are not rigorously defined concepts, despite decades of hard work by statisticians and philosophers of science. Regardless of the interpretation, Bayesian methods differ from frequentist in that they allow probability statements to be made about any unknown value, and not just those that represent eternally replicable random sampling.

Textbook examples often start with algebraically tractable posterior distributions, but in practice this is generally either infeasible or too time-consuming and prone to human error to be worthwhile. Instead, software allows the analyst to run one or more Markov chains of pseudorandom values which converge to a stationary distribution equivalent to draws from the joint posterior distribution of all the parameters of interest. From a large enough number of these draws, estimation and inference can be done empirically. The older algorithms, random walk Metropolis-Hastings and the Gibbs Sampler, take random steps through parameter space, and accept or reject the new location on the basis of its posterior probability.

This can work well under some circumstances but under others can require very large numbers of draws before they accurately represent the posterior distribution (convergence). Problems like this commonly arise when parameters are correlated (as, for example the intercept and slope of a bivariate linear regression is with only a few data), when priors are not ideal matches for the likelihood (a subtle topic beyond the scope of this paper but discussed in Bayesian textbooks [6]), or when initial values are poor guesses. Hamiltonian Monte Carlo addresses these issues by using Hamilton's equations of motion with periodic random impulses [7]. Exploration of the posterior probability is then analogous to a particle moving in a force field (picture a beachball rolling in the hollow between sand dunes, with occasional random kicks – gravity is the force providing the Hamiltonian motion), and as the joint posterior distribution guides movement to the region of highest posterior probability, the problems of sampling using random steps disappear. Even chains with poor initial values can still reveal the whole posterior distribution relatively quickly [7]. However, for a computer to perform this imitation of life, computationally expensive numerical integration and differentiation are required, but the lifting of the problems associated with random walk Metropolis-Hastings and Gibbs more than compensates for this. The No U-Turn Sampler (NUTS) is the algorithm implemented in Stan [8], which helps by automatically tuning the parameters of Hamiltonian Monte Carlo, achieving nearly optimal integration time in recent tests using CmdStan [9].

In the companion paper, we present a comparison of the efficiency of StataStan alongside `bayesmh` for an item-response model [5].

3 The stan and windowsmonitor commands

3.1 Objectives and development

Building on the history of linking Stata to WinBUGS [2], we sought to provide a single Stata command which would dispatch a specified Stan model code along with data. Because Stata can easily issue operating system commands, we use this to run the command-line implementation of Stan (CmdStan) and display summary results inside Stata. This is the approach also taken by the Stan interfaces from Matlab and Julia. CmdStan has to be installed before using StataStan but this is relatively straightforward with instructions on the Stan website, mc-stan.org.

We believe that Stata users who are becoming familiar with Bayesian techniques will find StataStan a usefully flexible, stable and fast tool. Also, people who already use Stata and Stan separately will find it helpful to keep everything in one workflow, because most people find it easier to work with one piece of software than to switch among them (and easier to maintain quality control). This allows data processing, simple analysis, complex modelling, graphics and writing out reports all in one place.

One unexpected problem we encountered was that Windows does not make its standard output on the command line available in such a way that Stata can display it in the results window until the external program has finished execution. In the case of complex Bayesian models which can take hours to run, this would be unacceptable. So, we wrote a small companion program called `windowsmonitor` which displays command-line output close to real time. `windowsmonitor` may provide a useful alternative to `shell` and `winexec` in other settings too.

3.2 The `stan` command for Stata

`stan` specifies what data are to be sent to CmdStan, with options controlling its settings and additional requirements such as sampling diagnostics or posterior modes. Data are passed to CmdStan in a text file, and outputs are returned similarly. These files are temporarily created in the CmdStan directory and then moved to the working directory; there is an option to retain them all, otherwise those that are rarely needed afterwards are deleted. Users should be mindful that any existing files in these locations with these names may be overwritten. A model has to be stored in its own file with extension `.stan`, and we discuss below different ways to achieve this.

3.3 Syntax of `stan`

`stan` *varlist* [*if in*] [, *options*]

3.3.1 Options

`datafile`(filename) specifies the name of a text file where `stan` will write the data on their way to Stan. This is done in the format used by R / S-plus and BUGS, for example with the auto dataset,

```
stan mpg,...
```

would write:

```
mpg=c(...)
```

The default datafile name is statastan_data.R

`modelfile`(filename) specifies the name of a text file, which must have the extension `.stan`, containing the Stan model. If this file already exists, the model is read from there, or it can be written into it using one of the methods detailed below under "Specifying the Stan model". The default is statastan_model.stan

`inline` instructs Stata to read the `.stan` model from a comment block inside the do-file (see below under 'Specifying the Stan model' for further discussion of `modelfile`, `inline` and `thisfile`).

`thisfile`(filename) specifies the name (and path, if required) of the current do-file; this is an option if inline has been specified (see below under

'Specifying the Stan model' for further discussion of `modelfile`, `inline` and `thisfile`).

`initsfile`(filename) specifies the name of a text file in R / S-plus format containing initial values. Because Stan is far less sensitive to initial values than software using older algorithms, we do not at present provide any mechanism like the datafile option to write this file from inside Stata.

`load` instructs Stata to read in the resulting draws as its current dataset

`diagnose`, if specified, will run Stan's diagnostics and display them after sampling, to examine whether the algorithm has run successfully.

`outputfile`(filename) provides the name for the text file into which CmdStan will write its outputs; the default is output.csv

`chainfile`(filename) provides the name for a comma-separated values (CSV) format file which will contain the draws from CmdStan; this is the same as `outputfile` but extra information is removed so it can be read into Stata using import delimited; the default is statastan_chains.csv

`mode` runs Stan's optimization to find posterior modes, and displays the results after sampling; it will also write the output into modesfile (see below)

`modesfile`(filename) provides the name of a text file to hold output from CmdStan's estimation of modes; the default is modes.csv

`winlogfile`(filename) provides the name of a temporary file to hold Windows output (see windowsmonitor); windowsmonitor will display this Stata's results window so there is no need we know of to change this from the default, which is winlog.txt

`seed`(integer) provides an integer psuedorandom number generator seed for Stan

`warmup`(integer) specifies the number of warmup draws, which are discarded from output and summaries

`iter`(integer) specifies the number of iterations (draws) to retain after warmup

`thin`(integer) specifies how much thinning of draws: if thin is set to n, Stan will retain one out of every n draws in output files and use the thinned draws for summaries; default is 1 (no thinning).

`chains`(integer) determines how many chains to run, in parallel if possible (regardless of the Stata flavor installed).

`skipmissing` will remove missing data observation-wise (on a cell-by-cell basis inside each column) before sending to Stan. This would apply if you want to send a series of vectors of different sizes by making these appear as 'variables' in your Stata data. This could be useful in the context of multilevel models, with smaller vectors of cluster-level data. It is not a natural way to think of Stata data so should be used with caution because it will apply to all the variables in *varlist*.

`matrices`(string) provides a list of matrices to send to Stan, or if set to "all", it will send all current matrices. These are written into the datafile as two-dimensional arrays.

`globals`(string) provides a list of global macros to send to Stan, or "all" to send all current global macros. These are written into the datafile as scalars.

Care should be taken not to write a string value as this will probably cause an error from CmdStan.

`keepfiles` instructs `stan` to keep all files produced along the way, otherwise, the model file, C++ file, executable file, chains file and (if produced) modes file will be retained in the working directory.

`stepsize`(integer) sets the stepsize for Hamiltonian Monte Carlo, default 1 (see the Stan manual for more detail [4])

`stepsizejitter`(integer) sets the stepsize jitter for Hamiltonian Monte Carlo, default 0 (see the Stan manual for more detail [4])

4 Specifying the Stan model

There are at least three ways of specifying the Stan model. Firstly, a .stan file can be written externally, for example in a text editor, and then named with the modelfile option. This has the disadvantage that updates to the analysis may require synchronised changes in the do-file and the model file. However, we recommend this as the starting point for new users of StataStan, because it avoids any bugs in writing and reading text files, and allows one to begin immediately using examples from the Stan manual and website. A second option is to include the code inside a comment block in the do-file. If the inline and thisfile options are used, Stata will read the text contents of thisfile, identify the comment block that begins (on the line following the `/*` symbol) with the word data:

```
/*
data{
int<lower=0> N;
int<lower=0,upper=1> y[N];
}
parameters {
real<lower=0,upper=1> theta;
}
model {
theta ~ beta(1,1);
y ~ bernoulli(theta);
}
*/
```

and write the contents of the block to the `.stan` file specified in `modelfile`.

The third option is to include the model code in the do-file and have Stata loop over its lines and write it to the `modelfile`:

```
    tempname writemodel
    file open `writemodel' using "mystanmodel.stan", ///
        write replace
```

```
#delimit ;
foreach line in
    "data { "
          "  int<lower=0> N; "
    "  int<lower=0,upper=1> y[N];"
    "} "
    "parameters {"
    "  real<lower=0,upper=1> theta;"
    "} "
    "model {"
    "  theta ~ beta(1,1);"
    "  y ~ bernoulli(theta);"
    "}"
    {;
    #delimit cr
    file write `writemodel' "`line'" _n
}
file close `writemodel'
stan y, modelfile("mystanmodel.stan") ///
    cmd("$cmdstandir") globals("N")
```

This has the advantage that all Stata and Stan code is in one file, but does not rely on naming or finding the do-file.

At present, the inline approaches (options two and three above) do not accommodate multiple blocks of code but we intend to add this capability.

4.1 The `windowsmonitor` program

`windowsmonitor` is a wrapper extending the ability of `shell`. It will be called by `stan` under Windows only, and it will return an error message if it is used in Mac or Linux computers. It intercepts the stdout stream (text that is displayed on the screen for command line programs) and prints it inside Stata. This is done by diverting stdout to a text file, checking that file every 2 seconds for new content and displaying that in Stata if it finds any. This continues until it receives a message that it is finished (in the form of a final line of output: ”Finished!”), which is added automatically. The only consideration for the user is to avoid using windowsmonitor to carry out any task that could write a single line ”Finished!” for any reason, because this will terminate the display inside Stata prematurely. If this is unavoidable, it is relatively simple to amend the signal word ”Finished!” in the source code.

4.1.1 Syntax of windowsmonitor

`windowsmonitor`, command() [*options*]

4.1.2 Options

`command`(string) contains the Windows command line code to be sent for execution `waitsecs`(integer) specifies the number of seconds to wait for output to appear before giving up; default 20. `winlogfile`(filename) specifies the file into which stdout should be diverted; default winlog.txt

`windowsmonitor` will also create a file called wmbatch.bat. If this survives execution, it can safely be deleted later.

5 Considerations for newcomers to Stan

Newcomers are strongly advised to work through some of the examples in the Stan manual before attempting serious applications. The Stan user must specify the type (such as integer or real number) as data or parameters. This allows Stan to make calculations efficient and helps with checking for inadvertent errors at compile time. Stan will translate the model to C++, which is itself a 'typed' language. For the most part, the Stata user need not be concerned with this, other than the obvious choice when writing the Stan code, but one potential pitfall is when reading in data from non-native file formats into Stata and sending it via `stan`. Floating-point precision means that what the human reads may not match what the computer stores, and this may lead to a "type mismatch" error message from CmdStan.

The statistics that are reported by CmdStan, and hence displayed by `stan`, are: the mean of draws from the posterior, the Monte Carlo Standard Error (MCSE) representing the uncertainty in the results arising from a finite number of draws, the standard deviation, 5th, 50th and 95th centiles of the draws, the number of effective independent samples (N_Eff, which accounts for autocorrelation in the chains) and number of effective independent samples obtained per second (N_Eff/s), and a measure of convergence (R_hat). The calculation of these measures is set out in [6]. N_Eff and R_hat are best assessed across multiple chains, so we advise users to run at least 4 chains as a general rule. `stan` can run parallel chains on multicore computers, even if Stata/MP is not installed, so most modern laptops can run 4 chains simultaneously. In the authors' experience, this runs in about half the time of serial chains.

Beyond these reported statistics, the value of loading the draws from the posterior distributions is that bespoke derived values can be calculated and summarised by the user inside Stata, to provide decision theoretic outputs. To give an example from health economics, a meta-analysis in Stan providing inference on effectiveness of alternative drugs can be loaded into Stata and then be combined with constant costs to derive a new cost-effectiveness variable, allowing probability statements about whether the cost-effectiveness exceeds a willingness-to-pay threshold. Another important benefit of working with the posterior draws is that the covariance structure among the parameters is preserved, while the tabulated summaries provide only marginal inferences.

Another consideration is that the number of available CPU cores needs to be specified when installing CmdStan itself, and the StataStan `chains()` option can only parallelise up to this number [10].

Stan model code allows for vectorised statements such as

`y ∼ bernoulli(theta);`

instead of

`for (n in 1:N) { y[n] ∼ bernoulli(theta); }`.

Both can be used in Stan, but the vectorised version is generally faster in execution.

6 Example

All the models set out in the Stan manual and website can be directly fitted using StataStan, including many that are not possible in `bayesmh`. We can use StataStan for a very simple example to estimate the probability of success θ in a Bernoulli process

$Pr(y_i) = \theta, 1 \leq i \leq 10, i \in \mathbb{N}$

when we have ten data: eight failures and two successes. We will apply a flat prior distribution over $[0, 1]$, either by explicitly specifying it or by omitting it because Stan uses uniform priors as default, provided that bounds on the parameter have been specified. The corresponding `bayesmh` command is:

```
bayesmh y, likelihood(dbernoulli({theta})) prior({theta},beta(1,
1))
```

The Stan code for this example is contained in the examples folder inside CmdStan.

```
data {
  int<lower=0> N;
  int<lower=0,upper=1> y[N];
}
parameters {
  real<lower=0,upper=1> theta;
}
model {
  theta ~ beta(1,1);
  y ~ bernoulli(theta);
}
```

Note that the code is arranged in blocks of: data, parameters and model. Other types of block can also be included, described in full in the Stan manual. Each object, whether data or parameter, used in the model must be declared with its type and any constraints before it can be used. Like BUGS and JAGS, the assignment operator $<-$ is used to calculate a value and store it in the object named on the left-hand side, while the $\sim$ operator has two functions. In the line `theta ~ beta(1,1);`

we are specifying a prior distribution (because theta is already declared as a parameter), and in the line

`y[n] ~ bernoulli(theta);`

we are incrementing the log-probability by the likelihood contribution of one observation according to the Bernoulli probability given the current estimate of theta.

Having specified this model, we can make the data:

```
clear
set obs 10
gen y=0
replace y=1 in 2
replace y=1 in 10
```

and then call `stan`:

```
quietly count
global N=r(N)
global cmdstandir "/path_to/CmdStan"
stan y, modelfile("bernoulli.stan") cmd("$cmdstandir")
    globals("N")
```

The first output to be displayed concerns translating the model to C++ and then compiling that. Compiling can be time-consuming, but does not have to be done again unless the model changes.

```
--- Translating Stan model to C++ code ---
bin\stanc.exe bernoulli.stan --o=bernoulli.hpp
Model name=bernoulli_model
Input file=bernoulli.stan
Output file=bernoulli.hpp

--- Linking C++ model ---
g++ -DBOOST_RESULT_OF_USE_TR1 -DBOOST_NO_DECLTYPE -DBOOST_DISABLE_ASSERTS -I src -I s
> tan_2.9.0/src -isystem stan_2.9.0/lib/stan_math_2.9.0/ -isystem stan_2.9.0/lib/stan
> _math_2.9.0/lib/eigen_3.2.4 -isystem stan_2.9.0/lib/stan_math_2.9.0/lib/boost_1.58.
> 0 -Wall -pipe -DEIGEN_NO_DEBUG -m32 -Wno-unused-function -Wno-uninitialized    -O3
> -o bernoulli.exe src/cmdstan/main.cpp -include bernoulli.hpp  -static-libgcc -stati
> c-libstdc++
```

After compilation is finished, we will see some settings for CmdStan:

```
method = sample (Default)
sample
        num_samples = 1000 (Default)
        num_warmup = 1000 (Default)
        save_warmup = 0 (Default)
        thin = 1 (Default)
adapt
        engaged = 1 (Default)
        gamma = 0.050000000000000003 (Default)
...
```

Then, we see the iterations appear, followed by a total time to do the sampling.

```
...
Iteration: 1600 / 2000 [ 80\%]  (Sampling)
```

```
Iteration: 1700 / 2000 [ 85\%]  (Sampling)
Iteration: 1800 / 2000 [ 90\%]  (Sampling)
Iteration: 1900 / 2000 [ 95\%]  (Sampling)
Iteration: 2000 / 2000 [100\%]  (Sampling)
#
#  Elapsed Time: 0.019 seconds (Warm-up)
#                0.045 seconds (Sampling)
#                0.064 seconds (Total)
#
```

This is followed by a summary of the parameters

```
                Mean     MCSE   StdDev     5%    50%    95%  N_Eff  N_Eff/s    R_hat
theta           0.24 6.7e-003 1.2e-001  0.082   0.23   0.46    300     6673 1.0e+000

Samples were drawn using hmc with nuts.
For each parameter, N_Eff is a crude measure of effective sample size,
and R_hat is the potential scale reduction factor on split chains (at
convergence, R_hat=1).
```

So, this shows us that the posterior mean for θ was 0.24 (pulled upward from the maximum likelihood estimate by the flat prior and the small dataset). If `mode` was specified, we will then see the posterior mode:

```
Log-probability at maximum: -5.004020214080811

---------------------
          | Posterior
Parameter |      Mode
----------+----------
theta     |   .200004
---------------------
```

which is directly comparable (with a flat prior) to the maximum likelihood estimate, 0.2.

If we specified `diagnose`, we will see corresponding output; the reader is referred to the Stan manual for details on this:

```
TEST GRADIENT MODE

Log probability=-7.10591

param idx           value           model     finite diff           error
        0       -0.557247        -1.37022        -1.37022   -1.66588e-010
```

Finally, if we specified `load`, we will see some Stata-generated summary, including the 95% credible interval:

```
    variable |         N      mean        sd  se(mean)       min        p1        p5
-------------+----------------------------------------------------------------------
       theta |      1000  .2485084  .1121162  .0035454   .019246  .0477189  .0814933
------------------------------------------------------------------------------------

     variable |       p25       p50       p75       p95       p99
 -------------+--------------------------------------------------
        theta |     .1628   .244064  .3222845  .4458295  .5513045
 ----------------------------------------------------------------
95\% CI for theta: .0656607497483492 to .4934002541005615
```

which is very similar to the approximate confidence interval:

```
. cii proportions 10 2, wilson

                                                         ------ Wilson ------
    Variable |        Obs  Proportion    Std. Err.       [95\% Conf. Interval]
-------------+---------------------------------------------------------------
             |         10          .2    .1264911        .0566822    .5098375
```

We also find our data replaced with variables called theta (which contains draws for the parameter of that name), lp__, accept_stat__, stepsize__, treedepth__, n_leapfrog__ and n_divergent__, all of which are created by CmdStan to track progress of the algorithm, and can be safely deleted unless needed for methodological investigations. The theta variable, containing the draws from the posterior, can then be used for graphics or further inference.

7 Conclusion

Stan continues to develop rapidly, with one major project being the inclusion of Riemann manifold Hamiltonian Monte Carlo which will provide further significant improvements in speed and stability [11]. StataStan can readily track this by adding new options which are passed to future versions of CmdStan.

Stan and all its interfaces have been made possible by enthusiastic contributions from developers around the world, co-ordinated by a core team. We encourage all interested Stata users to visit the website and to become involved there through reporting issues and suggesting improvements [4].

8 Acknowledgements

We thank the Institute of Education Sciences for partial support of this work. We are also grateful to users who have tested StataStan and provided feedback, as well as John Thompson of the University of Leicester and Charles Opondo of the University of Oxford for suggesting ways of in-line model specification.

References

[1] George Leckie & Chris Charlton. runmlwin - A Program to Run the MLwiN Multilevel Modelling Software from within Stata. *Journal of Statistical Software* (2013); 52 (11): 1–40.

[2] John Thompson. *WinBUGS from Stata.* http://www2.le.ac.uk/departments/health-sciences/research/gen-epi/Progs/winbugs-from-stata (Accessed 17 March 2016)

[3] StataCorp. `[BAYES]` manual, version 14.

[4] Stan Development Team. *Stan Modeling Language Users Guide and Reference Manual, Version 2.11.0*, 2016 http://mc-stan.org

[5] NOTE: PLEASE INSERT CROSS-REFERENCE TO THE COMPANION PAPER

[6] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari & Donald B Rubin. *Bayesian Data Analysis*, 3rd edition. CRC Press, 2013

[7] Radford Neal. "MCMC Using Hamiltonian Dynamics". In *Handbook of Markov Chain Monte Carlo*, eds Steve Brooks, Andrew Gelman, Galin Jones, Xiao-Li Meng. CRC Press, 2011.

[8] Matthew Hoffman & Andrew Gelman. The no-U-turn sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research* (2014); 15: 1593–1623.

[9] Michael Betancourt. Identifying the Optimal Integration Time in Hamiltonian Monte Carlo. ArXiv:1601.00225v1. http://arxiv.org/abs/1601.00225 (accessed 30 March 2016)

[10] Stan Development Team. *CmdStan Interface User's Guide, Version 2.11.0*, 2016 http://mc-stan.org

[11] Mark Girolami & Ben Calderhead. Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* (2011); 73(2): 123–214.