The Journal of Open Source Software

greta: simple and scalable statistical modelling in R

Nick Golding[1]

1 School of BioSciences, University of Melbourne

DOI: 10.21105/joss.01601

Software

- Review
- Repository
- Archive

Submitted: 26 July 2019
Published: 12 August 2019

Summary

Statistical modelling is used throughout the sciences. Often, statistical analyses require custom models that cannot be fitted using off-the shelf statistical software. These models can be specified in a statistical syntax and can then be automatically fit to data using methods such as Markov Chain monte Carlo (MCMC) and maximum likelihood. This lets users focus on the statistical nature of the model, rather than implementation details and inference procedures. Since the development of the widely successful WinBUGS (later developed as OpenBUGS; Spiegelhalter, Thomas, Best, & Lunn (2014)) a number of alternative software packages for custom statistical modelling have been introduced, including JAGS, Stan, and NIMBLE (Carpenter et al., 2017; de Valpine et al., 2017; Plummer & others, 2003). In these software packages, users typically write out models in a domain-specific language, which is then compiled into computational code. Though see the Python packages PyMC and Edward (Salvatier, Wiecki, & Fonnesbeck, 2016; Tran et al., 2016) in which models are specified in Python code.

With increasing quantities of data, complexity, and realism of statistical models that users wish to build with these software, there is a push for software that scales better with data size and model complexity. More recently, custom statistical modelling software has focussed on methods such as Hamiltonian Monte Carlo (rather than Gibbs samplers) in order to improve to computational efficiency. This can be seen for example in the development of Stan (Carpenter et al., 2017).

greta is an package for statistical modelling in R (R Core Team, 2019) that has three core differences to commonly used statistical modelling software packages:

1. greta models are written interactively in R code rather than in a compiled domain specific language.
2. greta can be extended by other R packages; providing a fully-featured package management system for extensions.
3. greta performs statistical inference using TensorFlow (Abadi et al., 2015), enabling it to scale across modern high-performance computing systems.

greta can be used to construct both Bayesian and non-Bayesian statistical models, and perform inference via MCMC or optimisation (for maximum likelihood or maximum *a posteriori* estimation). The default MCMC algorithm is Hamiltonian Monte Carlo, which is generally very efficient for Bayesian models with large numbers of parameters or highly-correlated posteriors.

The project website https://greta-stats.org/ hosts a *getting started* guide, worked examples of analyses using greta, a catalogue of example models, documentation, and a user forum.

The remainder of this paper provides an example of custom statistical modelling in greta, discusses the computational implementation of greta and how it can be extended, and highlights features that are planned for inclusion in the package.

Example

The following illustrates a typical modelling session with greta, using a Bayesian hierarchical model to estimate the treatment effect of epilepsy medication using data provided in the MASS R package (Venables & Ripley (2002), distributed with R) and analysed in the corresponding book.

Before we specify the greta model we format the data, adding a numeric version of the treatment type and making a vector of the (8-week) baseline counts for each subject (these counts are replicated in the epil object).

```
library(MASS)
epil$trt_id <- as.numeric(epil$trt)
baseline_y <- epil$base[!duplicated(epil$subject)]
```

Next we load greta and start building our model, starting with a random intercept model for the baseline (log-)seizure rates, to account for the fact that each individual will have a different seizure rate, irrespective of the treatment they receive. Variables in greta models - like `subject_mean`, and `baseline_effects` in the below code - are represented by `greta_array` objects, which have unknown values and are used to interactively build up the statistical formulation of the model (see *Implementation* for details).

```
library(greta)

# priors
subject_mean <- normal(0, 10)
subject_sd <- cauchy(0, 1, truncation = c(0, Inf))

# hierararchical model for baseline rates (transformed to be positive)
subject_effects <- normal(subject_mean, subject_sd, dim = 59)
baseline_rates <- exp(subject_effects)
```

Next we build a model for the effects (the ratio of post-treatment to pre-treatment seizure rates) of the two treatments: *placebo* and *progabide*. We give these positive-truncated normal priors (they are multiplicative effects, so must be positive), and centre them at 1 to represent a prior expectation of no effect. We multiply these effects by the baseline rates to get the post-treatment rates for each observation in the dataset.

```
treatment_effects <- normal(1, 1, dim = 2, truncation = c(0, Inf))
post_treatment_rates <- treatment_effects[epil$trt_id] *
  baseline_rates[epil$subject]
```

Finally we specify the distributions over the observed data. Here we use two likelihoods: one for the baseline count (over an 8 week period) and one for each of the post-treatment counts (over 2 week periods). We multiply our modelled weekly rates by the number of weeks the counts represent to get the appropriate rate for that period.

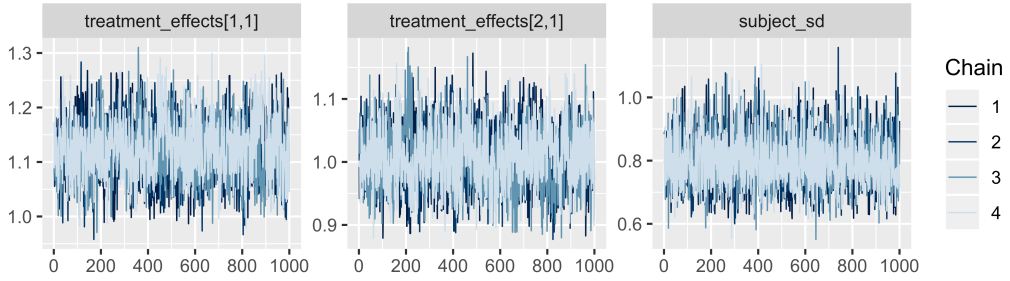
```
# likelihood
distribution(baseline_y) <- poisson(baseline_rates * 8)
distribution(epil$y) <- poisson(post_treatment_rates * 2)
```

Now we can create a model object using these greta arrays, naming the parameters that we are most interested in, and then run 4 chains of a Hamiltonian Monte Carlo sampler on the model, taking around 25 seconds on a laptop.

```
m <- model(treatment_effects, subject_sd)
draws <- mcmc(m, chains = 4)
```

The `draws` object contains posterior samples in an `mcmc.list` object from the coda package (Plummer, Best, Cowles, & Vines, 2006), for which there are many packages and utilities to summarise the posterior samples. Here we'll use the bayesplot package (Gabry & Mahr, 2018) to create trace plots for the parameters of interest, and the coda package to get $\hat{R}$ statistics to assess convergence of these parameters.

```
bayesplot::mcmc_trace(draws)
```




```
coda::gelman.diag(draws)
```

```
## Potential scale reduction factors:
##
##                          Point est. Upper C.I.
## treatment_effects[1,1]         1.01       1.04
## treatment_effects[2,1]         1.01       1.05
## subject_sd                     1.00       1.01
##
## Multivariate psrf
##
## 1.02
```

We can summarise the posterior samples to get the treatment effect estimates for placebo and progabide, the first and second elements of `treatment_effects`, respectively.

```
summary(draws)$statistics
```

```
##                               Mean         SD     Naive SE Time-series SE
## treatment_effects[1,1] 1.1211557 0.05246518 0.0008295473    0.001509931
## treatment_effects[2,1] 1.0074123 0.04566422 0.0007220147    0.001395410
## subject_sd             0.8003238 0.07867619 0.0012439797    0.001333331
```

These parameter estimates tell us the ratio of seizure rates during and before the treatment period for both the drug and placebo treatments. To calculate the effect of the drug relative to the placebo, we would take the ratio of the seizure rates between the drug treatment and the placebo treatment. We didn't include that term in our model, but fortunately there's no need to re-fit the model. greta's `calculate()` function lets us compute model quantities after model fitting.

```
# create a drug effect variable and calculate posterior samples
drug_effect <- treatment_effects[2] / treatment_effects[1]
drug_effect_draws <- calculate(drug_effect, draws)
summary(drug_effect_draws)$statistics
```

```
##           Mean             SD       Naive SE Time-series SE
##   0.9004330745   0.0574290367   0.0009080328   0.0016683350
```

`calculate()` can also be used for posterior prediction: we can reuse greta arrays for model parameters in combination with predictor variables to make a greta array for the predicted values of the new observations, then use `calculate()` to compute the posterior samples for these predictions. This means greta can be used in a predictive modelling workflow in which the data to predict to isn't available before model fitting, and without having to hand-code the predictions for all posterior samples.

Implementation

As in the example above, users of greta build up their models by creating and manipulating g `reta_array` objects representing parameters of other quantities in the model. `greta_arrays` behave like R's arrays, vectors and scalars, but with unknown values. greta extends a number of R's mathematical functions and other operations to work with greta arrays, so users can manipulate them as they would any other numeric object in R.

Internally, each of these greta arrays is represented by an R6 object (Chang, 2019), with information on the greta arrays from which they were created, or which re created with them. Together, these R6 objects constitute a directed acyclic graph (DAG), combining data, operations, variables, and probability densities. This DAG is then used to construct a function in TensorFlow code representing the joint density of the model. This core computational functionality, including optimisers and MCMC samplers, is provided by the TensorFlow and TensorFlow Probability Python packages (Abadi et al., 2015; Dillon et al., 2017), accessed from R via the tensorflow and reticulate R packages (Allaire & Tang, 2019; Ushey, Allaire, & Tang, 2019).

Parallelisation

Whereas most MCMC software packages enable parallelisation by running each MCMC chain to run on a separate CPU, greta's use of TensorFlow means it can parallelise MCMC on a single chain across an arbitrary number of CPUs. By installing the appropriate version of TensorFlow, greta models can also be run on Graphics Processing Units (GPUs). greta is also integrated with the future R package (Bengtsson, 2019) for remote and parallel processing, providing a simple interface to run inference for each chain of MCMC on a separate, remote machine. As a consequence, inference on greta models can be scaled up to make use of modern high-performance compute systems.

Extending greta

greta is not only designed to be extensible, but makes a deliberately distinction between the API for *users* who construct statistical models using existing functionality, and *developers* who add new functionality. Rather than letting users directly modify the inference target within a model (which can be dangerous if used incorrectly), new probability distributions and operations are created using a developer user interface, exposed via the `.internals` object. Once developed in this way, it becomes simple to distribute this new functionality to

other users via an R package that extends greta. Linking to the well established R package mechanism means that greta extensions automatically come with a fully-featured package management system, with tooling for development and distribution via CRAN or code sharing platforms.

Whilst anyone can write and distribute their own extension package, an aim of the greta project is to maintain a set of extension packages that meet software quality standards and are completely interoperable, in a similar way to the 'tidyverse' of R packages (Wickham, 2017) for data manipulation. These packages will be hosted on both the project GitHub organisation at https://github.com/greta-dev/ and on CRAN. There are currently a number of extensions in prototype form hosted on the GitHub organisation, including extensions to facilitate Gaussian process modelling (greta.gp), modelling dynamic systems (greta.dynamics) and generalised additive modelling (greta.gam).

Future work

greta is under active development, and a range of of new features will be added to the core package and extension packages in the near future. Two of the most significant expected changes are the ability to perform inference on discrete-valued parameters, and to include direct or approximate marginalisation of parameters as part of a model.

Discrete parameters

greta currently only handles models with exclusively continuous-valued parameters, since these models are compatible with the most commonly used optimisation routines and the efficient HMC sampler that is used by default. In the near future, greta will be extended to enable users to perform inference on models with discrete-valued parameters as required, in combination with the (typically less efficient) samplers with which these models are compatible.

Marginalisation

Many common statistical modelling approaches, such as hierarchical models, use unobserved *latent* variables whose posterior distributions must be integrated over in order to perform inference on parameters of interest. Whilst MCMC is a general-purpose method for marginalising these parameters, other methods are often better suited to the task in specific models. For example where those latent variables are discrete-valued and efficient samplers cannot be used, or when deterministic numerical approximations such as a Laplace approximation are more computationally-efficient. A simple user interface to specifying these marginalisation schemes within a greta model is planned. This will enable users to experiment with combinations of different inference approaches without the need delve into nuances of implementation.

Acknowledgements

I'd like to acknowledge direct contributions from Simon Dirmeier, Adam Fleischhacker, Shirin Glander, Martin Ingram, Lee Hazel, Tiphaine Martin, Matt Mulvahill, Michael Quinn, David Smith, Paul Teetor, and Jian Yen, as well as Jeffrey Pullin and many others who have provided feedback and suggestions on greta and its extensions. I would also like to thank Nick Tierney for comments and edits on this manuscript. greta was developed with support from both a McKenzie fellowship from the University of Melbourne, and a DECRA fellowship from the Australian Research Council (DE180100635).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Retrieved from http://tensorflow.org/

Allaire, J., & Tang, Y. (2019). *Tensorflow: R interface to 'tensorflow'*. Retrieved from https://CRAN.R-project.org/package=tensorflow

Bengtsson, H. (2019). *Future: Unified parallel and distributed processing in r for everyone*. Retrieved from https://CRAN.R-project.org/package=future

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., et al. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, *76*(1). doi:10.18637/jss.v076.i01

Chang, W. (2019). *R6: Encapsulated classes with reference semantics*. Retrieved from https://CRAN.R-project.org/package=R6

de Valpine, P., Turek, D., Paciorek, C., Anderson-Bergman, C., Temple Lang, D., & Bodik, R. (2017). Programming with models: Writing statistical algorithms for general model structures with nimble. *Journal of Computational and Graphical Statistics*, *26*(2), 403–413. doi:10.1080/10618600.2016.1172487

Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., et al. (2017). Tensorflow distributions. *arXiv preprint arXiv:1711.10604*.

Gabry, J., & Mahr, T. (2018). *Bayesplot: Plotting for bayesian models*. Retrieved from https://CRAN.R-project.org/package=bayesplot

Plummer, M., & others. (2003). JAGS: A program for analysis of bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing* (Vol. 124). Vienna, Austria.

Plummer, M., Best, N., Cowles, K., & Vines, K. (2006). CODA: Convergence diagnosis and output analysis for mcmc. *R News*, *6*(1), 7–11. Retrieved from https://journal.r-project.org/archive/

R Core Team. (2019). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). Probabilistic programming in python using PyMC3. *PeerJ Computer Science*, *2*, e55. doi:10.7717/peerj-cs.55

Spiegelhalter, D., Thomas, A., Best, N., & Lunn, D. (2014). OpenBUGS user manual, version 3.2.3. *MRC Biostatistics Unit, Cambridge*.

Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D., & Blei, D. M. (2016). Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*.

Ushey, K., Allaire, J., & Tang, Y. (2019). *Reticulate: Interface to 'python'*. Retrieved from https://CRAN.R-project.org/package=reticulate

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with s* (Fourth.). New York: Springer. Retrieved from http://www.stats.ox.ac.uk/pub/MASS4

Wickham, H. (2017). *Tidyverse: Easily install and load the 'tidyverse'*. Retrieved from https://CRAN.R-project.org/package=tidyverse