Stan

Stan is a programming language for specifying statistical models. It is most used as a MCMC sampler for Bayesian analyses. Markov chain Monte Carlo (MCMC) is a sampling method that allows you to estimate a probability distribution without knowing all of the distribution's mathematical properties. It is particularly useful in Bayesian inference because posterior distributions often cannot be written as a closed-form expression. To use Stan, the user writes a Stan program that represents their statistical model. This program specifies the parameters in the model along with the target posterior density. The Stan code is compiled and run along with the data and outputs a set of posterior simulations of the parameters. Stan interfaces with the most popular data analysis languages, such as R, Python, shell, MATLAB, Julia and Stata. We will focus on using Stan from within R, using the `rstan` and `rstanarm` packages.

`rstanarm`

`rstanarm` is a package that works as a front-end user interface for Stan. It allows R users to implement Bayesian models without having to learn how to write Stan code. You can fit a model in `rstanarm` using the familiar `formula` and `data.frame` syntax (like that of `lm()`). `rstanarm` achieves this simpler syntax by providing pre-compiled Stan code for commonly used model types. It is convenient to use but is limited to the specific "common" model types. If you need to fit a different model type, then you need to code it yourself with `rstan`.

The model fitting functions begin with the prefix `stan_` and end with the the model type. Some examples include `stan_glm()` and `stan_glmer()`. See here for a full list of `rstanarm` functions. The modeling functions have two required arguments:
– `formula`: A formula that specifies the dependent and independent variables (`y ~ x1 + x2`).
– `data`: A data-frame containing the variables in the formula.

Additionally, there is an optional `prior` argument, which allows you to change the default prior distributions.

`rstan`

The `rstan` package makes it easy to implement a Stan program into your R workflow. The `stan()` function reads and compiles your Stan code and fits the model on your dataset. The `stan()` function has two required arguments:
– `file`: The path of the `.stan` file that contains your Stan program.
– `data`: A named list providing the data for the model.

See here for a full list of all optional arguments.

Example

As a simple example to demonstrate how to specify a model in each of these packages, we'll fit a linear regression model using the `mtcars` dataset. Our dependent variable is `mpg` and all other variables are independent variables.

```
library(tidyverse)
library(rstan)
library(rstanarm)

mtcars %>%
  head()
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

```
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

First, we'll fit the model using `rstanarm`. For a linear regression we use the `stan_glm()` function.

```
fit_rstanarm <- stan_glm(
  mpg ~ .,
  data = mtcars,
  family = "gaussian"
)

summary(fit_rstanarm)
```

```
##
## Model Info:
##  function:     stan_glm
##  family:       gaussian [identity]
##  formula:      mpg ~ .
##  algorithm:    sampling
##  sample:       4000 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 32
##  predictors:   11
##
## Estimates:
##               mean   sd    10%   50%   90%
## (Intercept)  12.7   19.9 -12.2  12.6  38.2
## cyl          -0.1    1.1  -1.5  -0.1   1.3
## disp          0.0    0.0   0.0   0.0   0.0
## hp            0.0    0.0   0.0   0.0   0.0
## drat          0.8    1.7  -1.3   0.8   3.0
## wt           -3.6    2.0  -6.1  -3.5  -1.0
## qsec          0.8    0.8  -0.2   0.8   1.7
## vs            0.3    2.2  -2.4   0.3   3.0
## am            2.5    2.2  -0.2   2.5   5.3
## gear          0.6    1.5  -1.3   0.6   2.6
## carb         -0.3    0.9  -1.3  -0.3   0.8
## sigma         2.8    0.5   2.2   2.7   3.4
##
## Fit Diagnostics:
##            mean   sd   10%   50%   90%
## mean_PPD 20.1    0.7 19.2  20.1  21.0
##
## The mean_ppd is the sample average posterior predictive distribution of the
outcome variable (for details see help('summary.stanreg')).
##
## MCMC diagnostics
##               mcse Rhat n_eff
## (Intercept)   0.4  1.0  2910
## cyl           0.0  1.0  3034
## disp          0.0  1.0  1897
## hp            0.0  1.0  2546
## drat          0.0  1.0  3392
## wt            0.0  1.0  1942
## qsec          0.0  1.0  2602
## vs            0.0  1.0  3328
## am            0.0  1.0  3304
## gear          0.0  1.0  3217
## carb          0.0  1.0  2035
```

```
## sigma         0.0  1.0  1901
## mean_PPD      0.0  1.0  3862
## log-posterior 0.1  1.0  1045
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude
measure of effective sample size, and Rhat is the potential scale reduction
factor on split chains (at convergence Rhat=1).
```

The output shows parameter summaries including means, standard deviations, and quantiles. Additionally, it shows the MCMC diagnostic statistics Rhat and effective sample size. These statistics are important for assessing whether the MCMC algorithm has converged.

Next, we'll fit the same model using `rstan`. The following is the Stan code for our model, saved in a file named `mtcars.stan` (you can create a `.stan` file in RStudio or by using any text editor and saving the file with the extension `.stan`).

```
data {
  int N;   // number of observations
  int K;   // number of predictors
  matrix[N, K] X;   // predictor matrix
  vector[N] y;      // outcome vector
}
parameters {
  real alpha;           // intercept
  vector[K] beta;       // coefficients for predictors
  real sigma;  // error scale
}
model {
  y ~ normal(alpha + X * beta, sigma);  // target density
}
```

Stan code is structured within "program blocks". The three program blocks `data`, `parameters`, and `model` are required for every Stan model.

The `data` block is for the declaration of variables that are read in as data. In our case, we have our outcome vector (`y`) and our predictor matrix (`X`). When declaring a matrix or vector as a variable you are required to also specify the dimensions of the object. Therefore, we will also read in the number of observations (`N`) and number of predictors (`K`).

The variables declared in the `parameters` block are the variables that will be sampled by Stan. In the case of linear regression, the parameters of interest are the intercept term (`alpha`) and the coefficients for the predictors (`beta`). Additionally, there is the error term, `sigma`.

The `model` block is where the probability statements about the variables are defined. Here we specify that the target variable has a normal distribution with mean `alpha + X * beta` and standard deviation `sigma`. In this block you can also specify prior distributions for the parameters. By default, the parameters are given flat (non-informative) priors.

Additionally, there are optional program blocks: `functions`, `transformed data`, `transformed parameters`, and `generated quantities`. See here if you are interested in learning about these program blocks.

Next, we need to format our data in the way that the Stan program expects. The `rstan::stan()` function requires the data to be passed in as a named list, the elements of which are the variables that you defined in the `data` block. For this program, we create a list with the elements `N`, `K`, `X`, and `Y`.

```
predictors <- mtcars %>%
  select(-mpg)
```

```
stan_data <- list(
    N = 32,
    K = 10,
    X = predictors,
    y = mtcars$mpg
  )
```

Now that we have our Stan code and data ready, we pass them into the `stan()` function to fit the model.

```
fit_rstan <- stan(
  file = "mtcars.stan",
  data = stan_data
)

fit_rstan
```

```
## Inference for Stan model: mtcars.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean    sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## alpha     12.75    0.50 20.54 -27.90  -0.70  12.73  26.34  53.14  1697    1
## beta[1]   -0.11    0.03  1.14  -2.36  -0.85  -0.11   0.63   2.19  1918    1
## beta[2]    0.01    0.00  0.02  -0.02   0.00   0.01   0.03   0.05  1884    1
## beta[3]   -0.02    0.00  0.02  -0.07  -0.04  -0.02  -0.01   0.02  2342    1
## beta[4]    0.79    0.03  1.73  -2.76  -0.31   0.79   1.91   4.17  2706    1
## beta[5]   -3.78    0.05  2.02  -7.62  -5.10  -3.76  -2.51   0.21  1898    1
## beta[6]    0.81    0.02  0.78  -0.72   0.30   0.81   1.33   2.39  1876    1
## beta[7]    0.40    0.04  2.25  -4.12  -1.09   0.37   1.87   4.80  2813    1
## beta[8]    2.50    0.04  2.16  -1.64   1.06   2.50   3.92   6.74  3248    1
## beta[9]    0.63    0.03  1.63  -2.54  -0.43   0.60   1.71   3.86  2653    1
## beta[10]  -0.16    0.02  0.90  -1.92  -0.75  -0.17   0.42   1.55  1763    1
## sigma      2.82    0.01  0.47   2.08   2.49   2.76   3.09   3.92  1825    1
## lp__     -47.22    0.09  3.05 -54.21 -48.92 -46.81 -45.05 -42.51  1046    1
##
## Samples were drawn using NUTS(diag_e) at Tue Sep 08 10:24:01 2020.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

`rstan` outputs similar summary statistics to `rstanarm`, including means, standard deviations, and quantiles for each parameter. These results are similar but not exactly the same as the results from `rstanarm`. They are different because the statistics are calculated based on random sampling from the posterior.

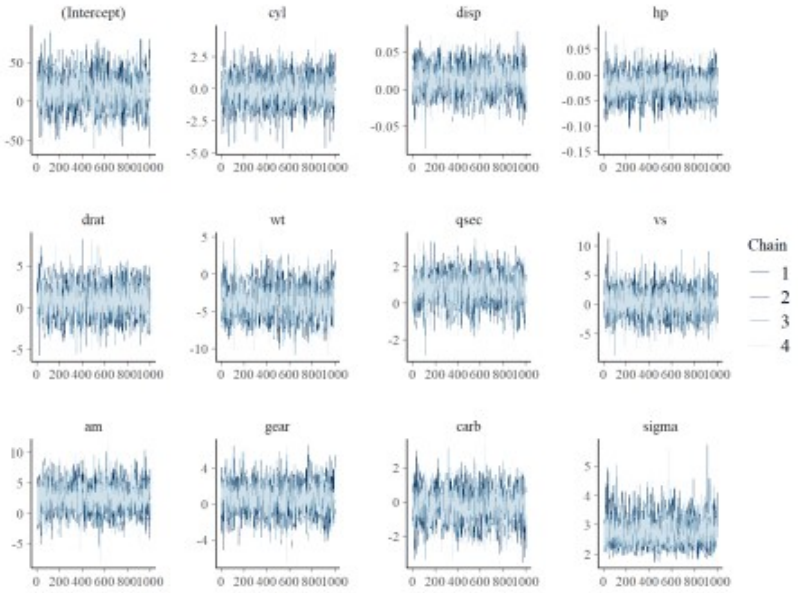
Assessing Convergence

When fitting a model using MCMC, it is important to check if the chains have converged. We recommend the `bayesplot` package to visually examine MCMC diagnostics. The `bayesplot` package supports model objects from both `rstan` and `rstanarm` and provides easy to use functions to display MCMC diagnostics. We will demonstrate the `mcmc_trace()` function to create a trace plot and the `mcmc_rhat()` function to create a plot of the Rhat values.

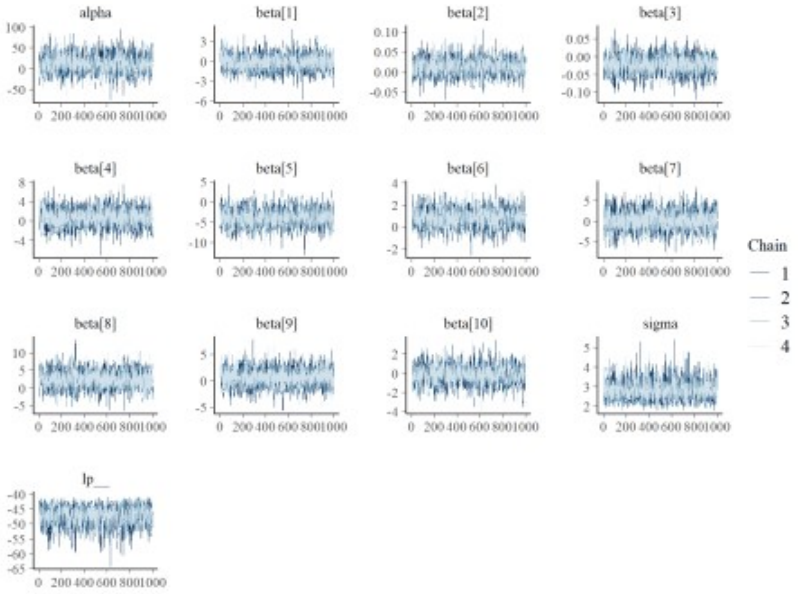
First, let us create trace plots using `mcmc_trace()`. A trace plot shows the sampled values of the parameters over the MCMC iterations. If the model has converged, then the trace plot should look like a random scatter around a mean value. If the chains are snaking around the parameter space or if the chains converge to different values, then that is evidence of a problem. We demonstrate the function using our model fits from both `rstanarm` and `rstan`.

```
library(bayesplot)

fit_rstanarm %>%
  mcmc_trace()
```



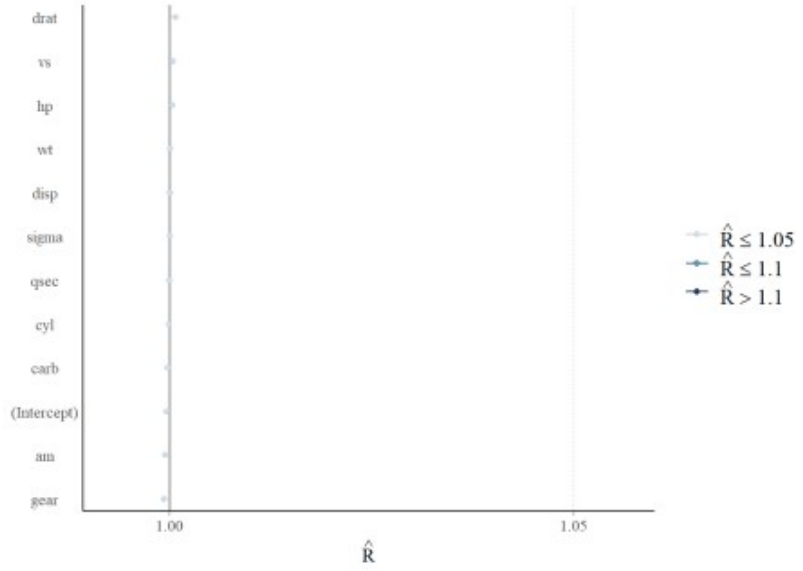

```
fit_rstan %>%
  mcmc_trace()
```



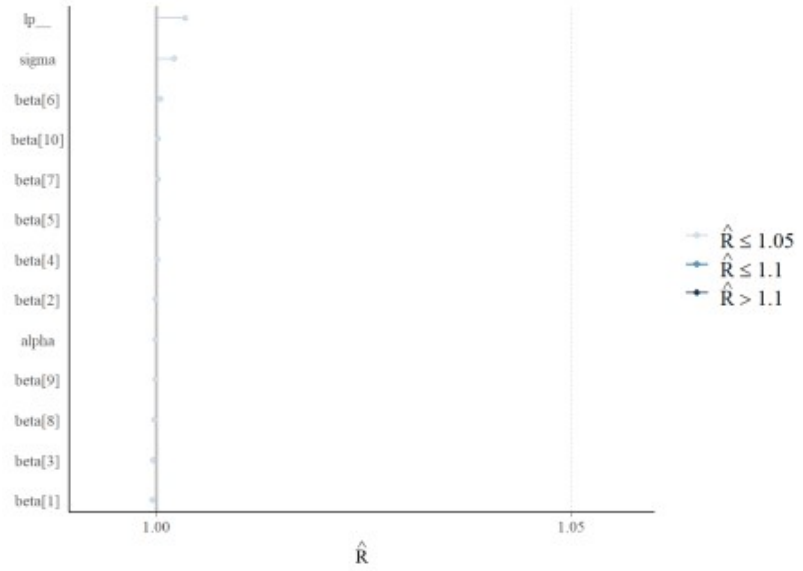
These trace plots suggest that both models have converged. For all parameters, the four chains have mixed and there are no clear trends.

Next, we'll examine the Rhat values using `mcmc_rhat()`. Rhat is a convergence diagnostic which compares parameter estimates across the chains. If the chains have converged and mixed well, then the Rhat value should be near 1. If the chains have not converged to the same value, then the Rhat value will be larger than 1. Rhat values of 1.05 or higher suggest a convergence issue. The `mcmc_rhat()` function requires a vector of Rhat values as an input, so we first extract the Rhat values using the `rhat()` function.

```
fit_rstanarm %>%
  rhat() %>%
  mcmc_rhat() +
  yaxis_text()
```




```
fit_rstan %>%
  rhat() %>%
  mcmc_rhat() +
  yaxis_text()
```



All Rhat values are below 1.05, suggesting that there are no convergence issues.

Hopefully, this provides a good starting point for building Stan models in R. Stan is a powerful tool for building Bayesian models, and these packages make it easy for R users to use Stan.