Stan Tutorial

Lauren Cappiello

2/8/2018

Installing Stan

Prerequisites: R (or RStudio)

Toolchain: Stan requires a toolchain to compile and run C++ code through R.

Configuration: get R ready to install Stan

Installing Stan: the RStan package

The following links will take you through the appropriate methods for your operating system:

- Installing RStan on Mac or Linux
- Installing RStan on Windows

Not an R user?

Loading Stan in R

The loading message will suggest some parallel processing options. Be aware that the default is to use all of your computer's cores. These lines of code are included (commented out).

```
library("rstan")
#rstan_options(auto_write = TRUE)
#options(mc.cores = parallel::detectCores())
```

Stan Code

Data Types

- Basic types: real, int, vector, row_vector, matrix
- Constrained types: simplex, unit_vector, ordered, positive_ordered, corr_matrix, cov_matrix

Bounded Variables

- Can put upper and lower bounds on variables
- Ex: Let σ^2 = sigsq
- real<lower=0> sigsq;

Program Blocks

- data
- transformed data
- parameters
- transformed parameters
- model
- generated quantities

The only required program block is the model block.

Limitations

- No discrete parameters
- Missing data must be coded as parameters
- C++ template code can be difficult to work with
- Limited data types and constraints
- Sampling is relatively slow
- Some models may not work in Stan or may require extensive reparameterization
- "Black box" modeling

So what is Stan doing?

Stan uses gradient-based MCMC for Bayesian inference, gradient-based variational methods for approximate Bayesian inference, and gradient-based optimization for penalized MLE.

. . . but it's difficult to determine the details of what Stan does within these frameworks.

Eight Schools Example

This example can also be found in the "getting started" section for RStan.

- SAT scores from eight high schools with some estimated treatment effect and estimated standard error for each treatment effect
- The model of interest is $y_j = \theta + error = \mu + \tau\eta_j + error$, $j = 1, ..., 8$.
- $\eta \sim N(0, 1)$ and $y \sim N(\theta, \sigma^2)$ where σ is the standard error of the effect measurements. This is included in the data.
- The following code will estimate μ, τ, and η_j (as well as $\theta_j = \mu + \tau\eta_j$).

We start by writing Stan code as a character string. Note: You may instead want to create separate Stan files. As the Stan input gets more complex, this is preferable to character strings.

```
eightschools <- "
data {
    int<lower=0> J;           // number of schools
    real y[J];                // estimated treatment effects
    real<lower=0> sigma[J];   // s.e. of effect estimates
}
parameters {
    real mu;                  // mean effect for schools
    real<lower=0> tau;        // variance
    real eta[J];              // individual school effect
}
transformed parameters {    // theta is a function of our parameters
    real theta[J];
    for (j in 1:J)
        theta[j] = mu + tau * eta[j];
}
model {
    target += normal_lpdf(eta | 0, 1);        // eta ~ N(0,1)
    target += normal_lpdf(y | theta, sigma);  // y ~ N(theta, sigma^2), theta(mu, tau, eta)
}"
```

First we input everything that appears in our data block as a list and then pass it to our `stan` function.

```
require("rstan")
set.seed(0)
schools_dat <- list(J = 8,
                    y = c(28,  8, -3,  7, -1,  1, 18, 12),
                    sigma = c(15, 10, 16, 11,  9, 11, 10, 18))
fit <- stan(model_code = eightschools,
            data = schools_dat, iter = 10000, warmup = 100, chains = 4)
```

The stan Function

`stan` function arguments

- `model_code`: the character string that contains your Stan code
- `file`: the filepath to a .stan file containing your Stan code
- `data`: the data to be passed to the `data` block
- `chains`: the number of Markov chains to run (default is 4)
- `iter`: the number of iterations for each chain (includes warmup)
- `warmup`: the number of warmup iterations per chain

The `stan` function can also handle previous fits, parameters of interest, instructions for thinning, initial values, and a seed. See the `stan` help file for details.

Results

Printing the fit will automatically give the estimated mean, standard error of the mean, standard deviation, and a number of percentiles for each estimated paramter. Stan also reports an effective sample size and how well the chains are mixing ($\hat{R}$ close to 1 indicate good mixing).

```
print(fit)
```

```
## Inference for Stan model: fa585c379601bb55be0139f77da54458.
## 4 chains, each with iter=10000; warmup=100; thin=1;
## post-warmup draws per chain=9900, total post-warmup draws=39600.
##
##            mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## mu         7.89    0.10 5.22  -2.44   4.59   7.87  11.18  18.30  2781    1
## tau        6.60    0.06 5.53   0.24   2.50   5.28   9.23  20.47  9153    1
## eta[1]     0.40    0.00 0.93  -1.47  -0.21   0.42   1.03   2.20 39600    1
## eta[2]     0.00    0.00 0.87  -1.73  -0.58   0.00   0.56   1.73 39600    1
## eta[3]    -0.19    0.00 0.93  -1.99  -0.82  -0.20   0.42   1.66 39600    1
## eta[4]    -0.03    0.00 0.88  -1.77  -0.61  -0.03   0.55   1.72 39600    1
## eta[5]    -0.35    0.00 0.88  -2.04  -0.94  -0.37   0.21   1.45 39600    1
## eta[6]    -0.21    0.00 0.88  -1.93  -0.79  -0.23   0.36   1.58 39600    1
## eta[7]     0.35    0.00 0.89  -1.48  -0.22   0.37   0.94   2.04 39600    1
## eta[8]     0.06    0.00 0.93  -1.78  -0.56   0.06   0.69   1.90 39600    1
## theta[1]  11.41    0.08 8.33  -2.19   5.99  10.32  15.63  31.57 10589    1
## theta[2]   7.88    0.06 6.30  -4.77   3.98   7.88  11.78  20.61 12324    1
## theta[3]   6.11    0.07 7.74 -11.38   1.99   6.65  10.88  20.36 11049    1
## theta[4]   7.63    0.06 6.56  -5.88   3.64   7.68  11.67  20.88 11993    1
## theta[5]   5.06    0.06 6.44  -9.07   1.24   5.54   9.43  16.43 12569    1
## theta[6]   6.11    0.06 6.70  -8.57   2.25   6.49  10.43  18.42 11990    1
## theta[7]  10.66    0.06 6.79  -1.27   6.10  10.06  14.54  25.99 11509    1
```

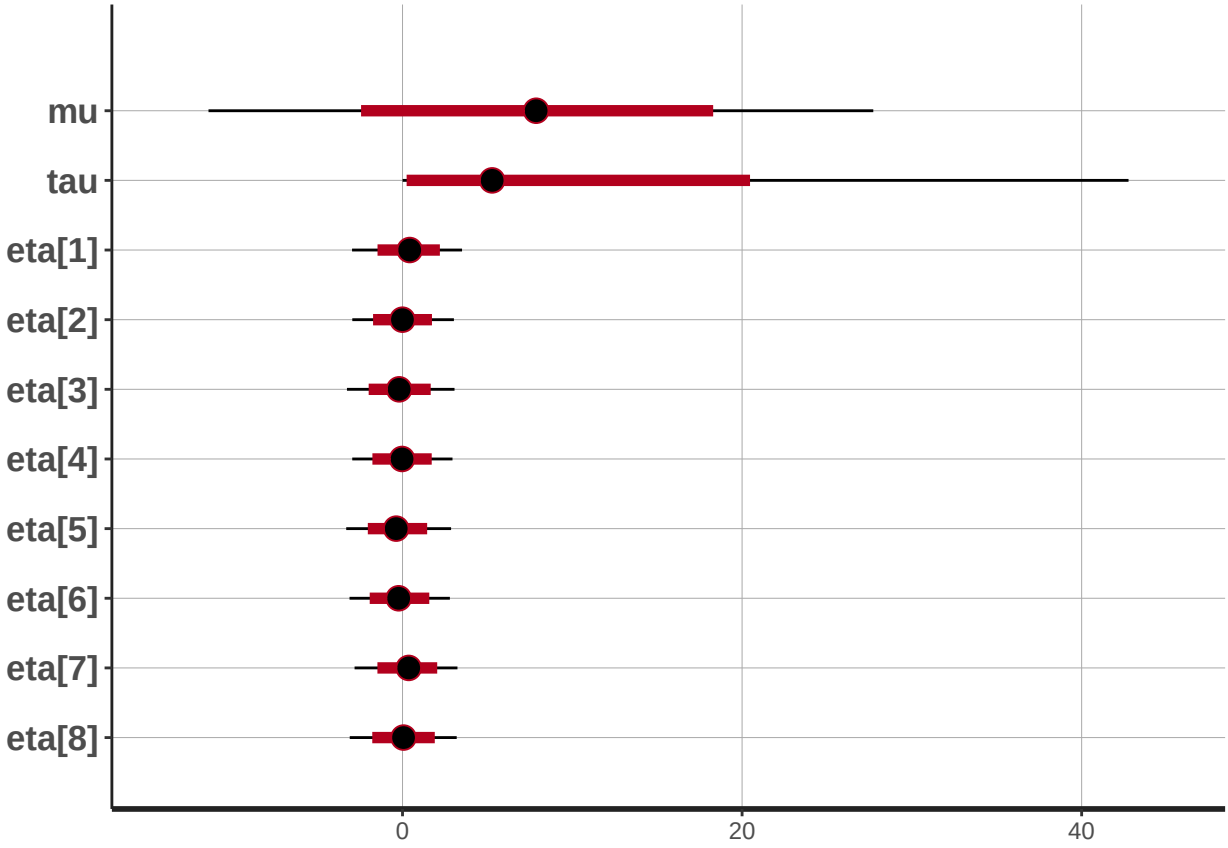
```
## theta[8]   8.46    0.08 7.96  -7.35   3.88   8.24  12.78  25.72  9847    1
## lp__     -39.53    0.03 2.64 -45.34 -41.13 -39.26 -37.66 -35.11 11125    1
##
## Samples were drawn using NUTS(diag_e) at Wed Feb  7 18:25:48 2018.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
#summary(fit)
```

Plotting the fit gives default 80% and 95% confidence intervals for each parameter. These confidence levels can be adjusted based on user preference. I have set the main confidence interval to 95% and the outer interval to 99.9%.

```
plot(fit, ci_level = 0.95, outer_level = 0.999)
```

```
## 'pars' not specified. Showing first 10 parameters by default.
## ci_level: 0.95 (95% intervals)
## outer_level: 0.999 (99.9% intervals)
```



Diasorin Example

```
diasorin <- "
data {
    int<lower=0> n1;  //n low
    int<lower=0> n2;  //n high
    real low[n1];     //low data of length n1
    real norm[n2];    //high data of length n2
}
parameters {
    real<lower=0> tau1;
    real<lower=0> tau2;
    real mu1;
    real mu2;
}
transformed parameters{
    real t1;
    real t2;
    t1 = 1.0/(tau1^(0.5));    //Stan uses standard deviation
    t2 = 1.0/(tau2^(0.5));
}
model {
    mu1 ~ normal(4.87, 0.05366563);    //Priors
    mu2 ~ normal(5.39, 0.05291503);
    tau1 ~ gamma(1.0376, 0.001);
    tau2 ~ gamma(1.0465, 0.001);
    low ~ normal(mu1, t1);            //Target
    norm ~ normal(mu2, t2);
}"
```

Pull Diasorin Stan Code Into R

```
set.seed(1)
dias.dat <- list(n1 = 19, n2 = 15,
                 low =
                     log(c(91, 46,  95,  60, 33, 410, 105, 43, 189,1097, 54,178, 114, 137, 233, 101, 25
                 norm = log(c(370, 267, 99,157, 75,1281, 48, 298, 268, 62,804,430,171,694,404)))
fit2 <- stan(model_code = diasorin,
            data = dias.dat, iter = 10000, warmup = 100, chains = 4)
```

Results

```
print(fit2)
```

```
## Inference for Stan model: 777482c65ceb8ab2b52f9b17f2dc9f9b.
## 4 chains, each with iter=10000; warmup=100; thin=1;
## post-warmup draws per chain=9900, total post-warmup draws=39600.
##
##        mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## tau1   1.28    0.00 0.39   0.62   0.99   1.24   1.52   2.14 18739    1
## tau2   1.29    0.00 0.44   0.57   0.97   1.24   1.55   2.29 15624    1
```

```
## mu1    4.86    0.00 0.05   4.76   4.82   4.86   4.89   4.96 39600    1
## mu2    5.40    0.00 0.05   5.29   5.36   5.39   5.43   5.50 39600    1
## t1     0.92    0.00 0.15   0.68   0.81   0.90   1.00   1.27 16579    1
## t2     0.92    0.00 0.17   0.66   0.80   0.90   1.01   1.32 13924    1
## lp__ -16.33    0.01 1.43 -19.92 -17.03 -16.01 -15.28 -14.55 13737    1
##
## Samples were drawn using NUTS(diag_e) at Wed Feb  7 18:26:57 2018.
## 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).
```

```
#summary(fit2)
```

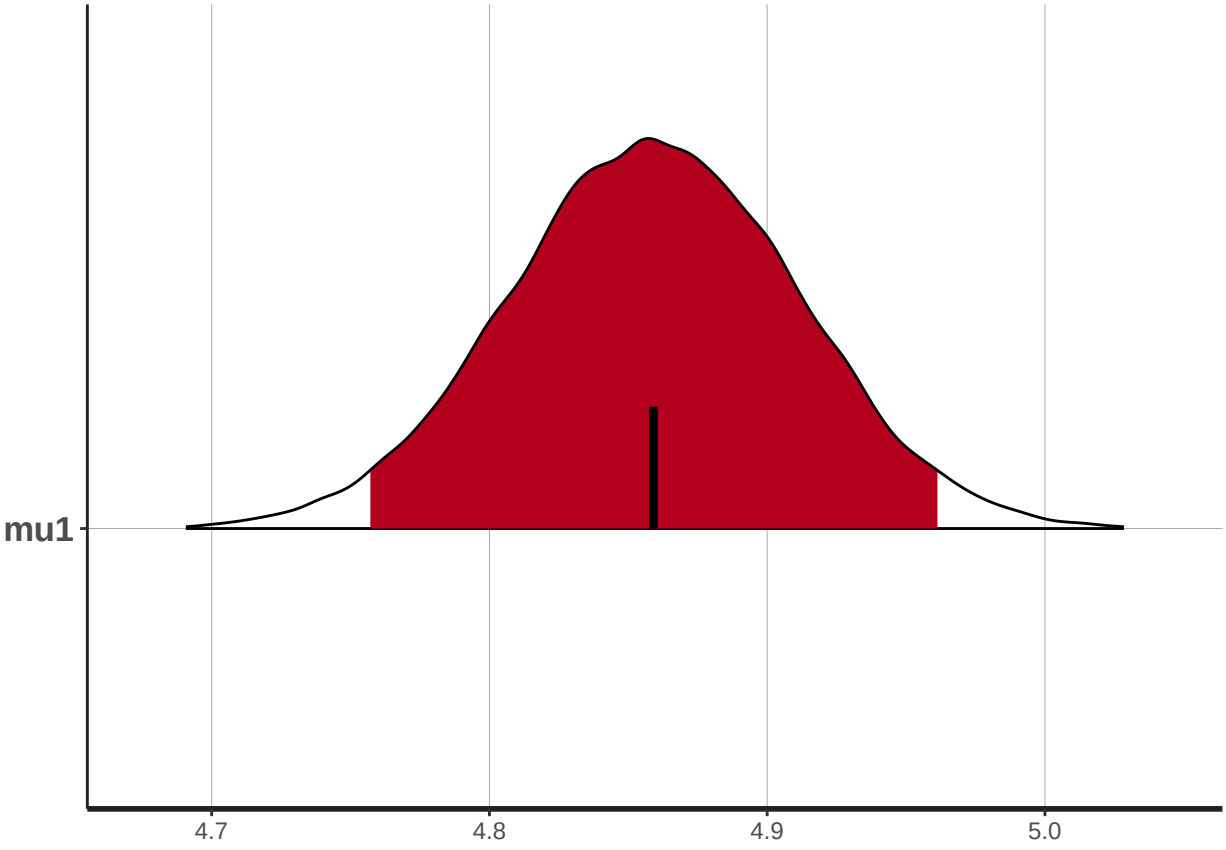
`show_density` has the `plot` function give the density of the estimated parameters instead of just a confidence interval.

The density of μ_1 based on the simulation:

```
plot(fit2, show_density=TRUE, pars="mu1", ci_level=0.95, outer_level=0.999)
```

```
## ci_level: 0.95 (95% intervals)
```

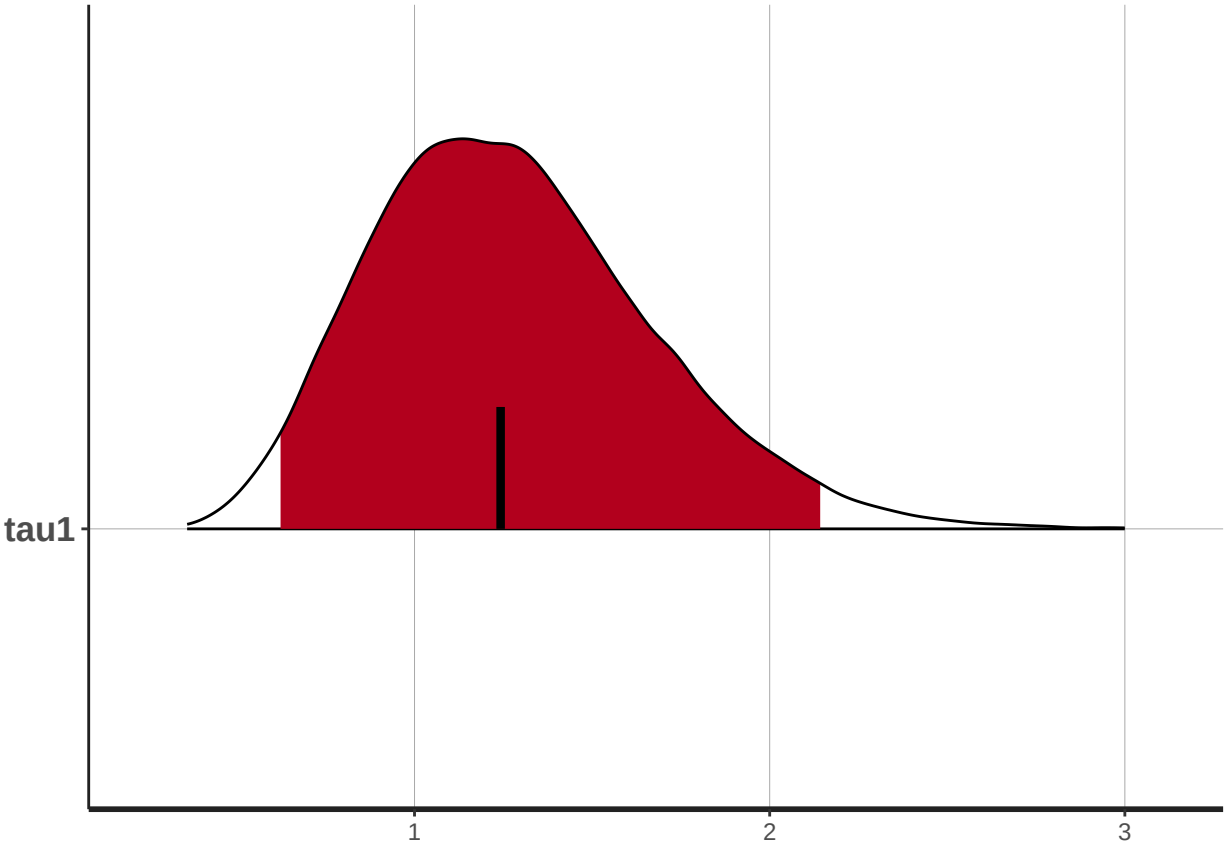
```
## outer_level: 0.999 (99.9% intervals)
```



The density of τ_1 based on the simulation:

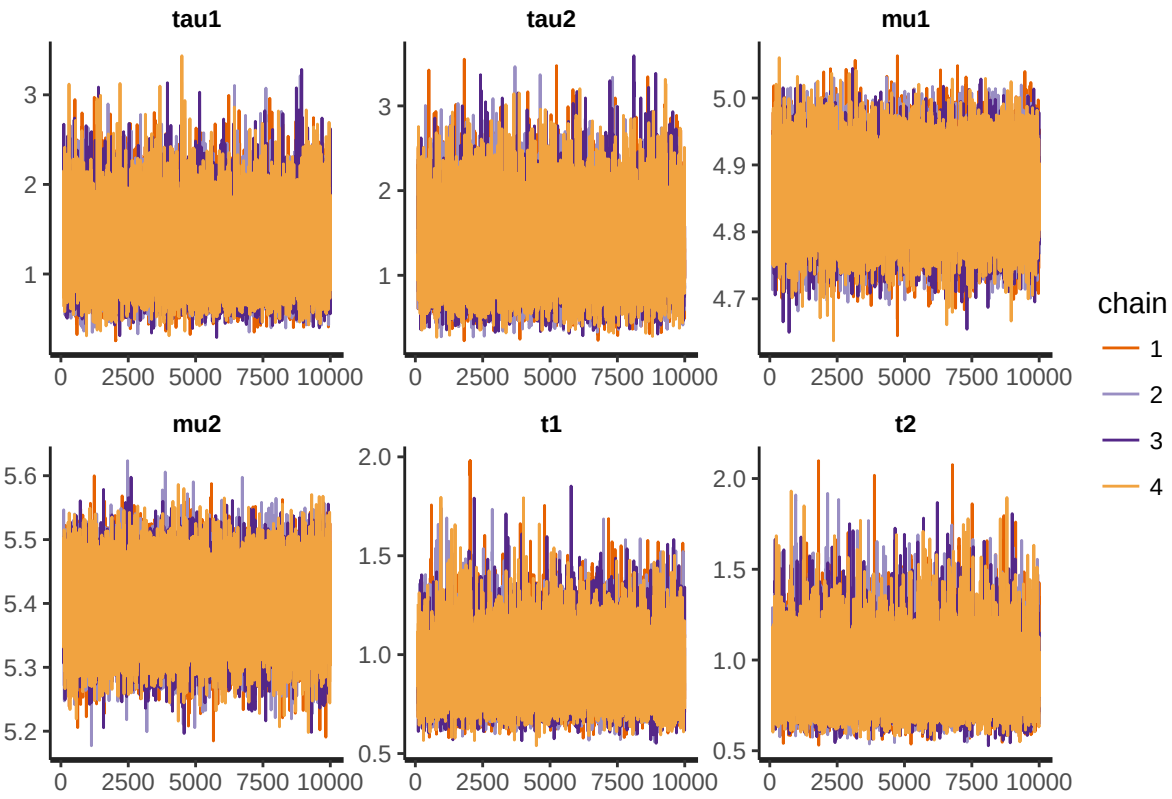
```
plot(fit2, show_density=TRUE, pars="tau1", ci_level=0.95, outer_level=0.999)
```

```
## ci_level: 0.95 (95% intervals)
## outer_level: 0.999 (99.9% intervals)
```



Trace plots of the fit show all four chains overlaid for each parameter. This should reflect our $\hat{R}$ values from our fit summary.

```
traceplot(fit2)
```



Indeed, the chains appear to be mixing well for all parameters.

Additional Information About Stan

More information about Stan

Stan Modeling Language User's Guide and Reference Manual

"Introduction to Bayesian Data Analysis and Stan with Andrew Gelman" (and the corresponding slides), a presentation by one of Stan's creators and core developers. Dr. Gelman discusses an example examining World Cup soccer rankings, another using geometry to model golf ball trajectories, and finally an example on relative birth rates per day of the year.