Introduction to Stan

Math 341

2019-03-22

Stan Example

Describing the model

```
data {
  int<lower=0> N;  // N is a non-negative integer
  int y[N];          // y is a length-N vector of integers
}
parameters {
  real<lower=0,upper=1> theta;  // theta is between 0 and 1
}
model {
  theta ~ beta (1,1);
  y ~ bernoulli(theta);
}
```

The chunk header for the chunk above looks like this:

```
{stan output.var = "simple_stan", cache = TRUE}
```

```
* `stan`: This is a Stan model, not R code.
* `output.var`: Name of R object for storing the stan DSO.
* `cache = TRUE`: Don't recompile this unless something changes.
```

Exercises 1 and 2

1. Identify as many differences as you can between this Stan code and the corresponding JAGS code.
2. What is this model? Draw a model diagram for this model. What sort of data must you provide?

Generating posterior samples

```
simple_stanfit <-
  sampling(
    simple_stan,
    data  = list(
      N = 50,
      y = c(rep(1, 15), rep(0, 35))
    ),
    chains = 3,      # default is 4
    iter = 1000,     # default is 2000
    warmup = 200     # default is half of iter
  )
```

Exercise 3

This looks pretty similar to `jags()`, but it isn't identical. List as many differences as you can.

Running `sampling()` produces a lot of output (hidden here with `results = "hide"` in the R chunk header). Here's just a portion of what is produced.

```
SAMPLING FOR MODEL 'stan-126b674f2d49f' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 1.7e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:  Elapsed Time: 0.006157 seconds (Warm-up)
Chain 1:                0.021385 seconds (Sampling)
Chain 1:                0.027542 seconds (Total)
```

```
simple_stanfit
```

```
## Inference for Stan model: 83dbe9f99dbf55ff04494fdddf566a3d.
## 3 chains, each with iter=1000; warmup=200; thin=1;
## post-warmup draws per chain=800, total post-warmup draws=2400.
##
##         mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## theta   0.31    0.00 0.07   0.19   0.26   0.30   0.35   0.45   838    1
## lp__  -32.63    0.02 0.74 -34.77 -32.78 -32.34 -32.15 -32.10  1279    1
##
## Samples were drawn using NUTS(diag_e) at Fri Mar 22 11:48:36 2019.
## 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).
```

Now what?

- Use `CalvinBayes::posterior()` to create a dataframe with posterior samples. These can be plotted or explored using `ggformula` or other familiar tools.
- Use `as.matrix()` or `as.mcmc.list()` to create an object that can be used with `bayesplot` just as we did when we used JAGS.

```
gf_dens(~theta, data = posterior(simple_stanfit))
mcmc_areas(as.mcmc.list(simple_stanfit), prob = 0.9, pars = "theta")
mcmc_trace(as.matrix(simple_stanfit))
```