Stan Hands-On Introduction

Ben Goodrich

Stan-Users NYC Meetup: June 24, 2014

Outline

Plan for Today

1. Install some flavor of Stan on everyone's laptop
2. Briefly talk about Bayesian inference and MCMC
3. Overview of the Stan modeling language
4. Examples of using Stan

Links

- http://mc-stan.org/ has everything
- Google Groups:
  - low-volume release announcements: https://groups.google.com/forum/?fromgroups#!forum/stan-announce
  - for help with your models / configuration problems: https://groups.google.com/forum/?fromgroups#!forum/stan-users
  - if you are interested in contributing to Stan: https://groups.google.com/forum/?fromgroups#!forum/stan-dev
- There is a Stan tag on StackOverflow and affiliates, but it is not used very much
- http://www.stat.columbia.edu/∼gelman/book/ Gelman's textbook, which also has links to lecture slides

Flavors of Stan

- “Stan” is a catch-all term that includes
  - libstan, a library for statistics and optimization
  - stanc, a parser for the Stan language
  - interfaces to libstan and stanc
    - CmdStan, for use via a command-line shell
    - RStan, for use via the R language
    - PyStan, for use via the Python language
    - MStan (MATLAB), StataStan (Stata), etc. are in progress
  - the Stan community
- Install the interface that is most comfortable for you

PyStan

- Requires Python 2.7+; see https://pystan.readthedocs.org/
- Helps to have matplotlib
- Windows: https://pystan.readthedocs.org/en/latest/windows.html
- Linux or OS X prior to Mavericks: Use pip via shell

```
sudo pip install pystan
```

- OS X Mavericks: Install from source via Terminal
    - Requires Cython and NumPy

```
wget https://github.com/stan-dev/pystan/archive/2.2.0.1.zip
unzip 2.2.0.1.zip
cd pystan-2.2.0.1
sudo export MAKEFLAGS = "-j4" # or another number besides 4
sudo \
ARCHFLAGS=-Wno-error=unused-command-line-argument-hard-error-in-future
\
sudo python setup.py install
export \
ARCHFLAGS=-Wno-error=unused-command-line-argument-hard-error-in-future
cd ..
```

RStan

- Not on CRAN yet
- Somewhat involved installation procedure
- https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
- Can utilize Amazon EC2 http://www.louisaslett.com/RStudio_AMI/ but “micro” instances have to little RAM

CmdStan

- Possible to access Stan from command-line
- Probably mainly of interest to potential developers

```
git clone https://github.com/stan-dev/cmdstan
make /path/to/stanfile-without.stan-extension
```

Bayes' Theorem

- Let $\overrightarrow{\theta}$ be a vector of unknown parameters
- Let $f(\cdot)$ be a PDF (continuous) or PMF (discrete)
  - Continuous example: Normal PDF is $f(y|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2\right)$
  - Discrete example: Poisson PMF is $f(y|\lambda) = \frac{\lambda^y \exp(-\lambda)}{y!}$
- The axioms of probability imply that

$$f\left(\overrightarrow{\theta}\middle|\text{data}\right) f(\text{data}) = f\left(\overrightarrow{\theta},\text{data}\right) = f\left(\overrightarrow{\theta}\right) f\left(\text{data}|\overrightarrow{\theta}\right)$$

$$\underbrace{f\left(\overrightarrow{\theta}\middle|\text{data}\right)}_{\text{posterior}} = \frac{\overbrace{f\left(\overrightarrow{\theta}\right)}^{\text{prior}}\overbrace{f\left(\text{data}|\overrightarrow{\theta}\right)}^{\text{likelihood}}}{\underbrace{f(\text{data})}_{\text{evidence}}}$$

Computation

- Easy to write down the right-hand side of a posterior distribution: $f\left(\overrightarrow{\theta}\middle|\text{data}\right) = \frac{f\left(\overrightarrow{\theta}\right)f\left(\text{data}|\overrightarrow{\theta}\right)}{f(\text{data})}$
- But hard to do anything with it analytically
  - $f(\text{data}) = \int_\Omega f\left(\overrightarrow{\theta}\right) f\left(\text{data}|\overrightarrow{\theta}\right) d\overrightarrow{\theta}$ is hard
  - Even if you do that, $f\left(\overrightarrow{\theta}\middle|\text{data}\right)$ is multidimensional so it is hard to integrate out a bunch of parameters to obtain the marginal distribution of interest
- Relatively recently, Markov Chain Monte Carlo (MCMC) has become a popular solution to these problems
- We can randomly draw from $f\left(\overrightarrow{\theta}\middle|\text{data}\right)$, even if we are unable to manipulate it analytically
- Draws are identically distributed but not independent

Two General MCMC Engines

- Metropolis-Hastings algorithm. Initialize $\overrightarrow{\theta}$ and repeat:
  1. Randomly draw a new parameter vector, $\overrightarrow{\theta}^*$, from some jumping distribution $q\left(\overrightarrow{\theta}^* \middle| \overrightarrow{\theta}, \text{data}\right)$
  2. Evaluate
  $$\frac{f\left(\overrightarrow{\theta}^* \middle| \text{data}\right)}{f\left(\overrightarrow{\theta} \middle| \text{data}\right)} \times \frac{q\left(\overrightarrow{\theta} \middle| \overrightarrow{\theta}^*, \text{data}\right)}{q\left(\overrightarrow{\theta}^* \middle| \overrightarrow{\theta}, \text{data}\right)} = \frac{f\left(\overrightarrow{\theta}^*\right) f\left(\text{data} \middle| \overrightarrow{\theta}^*\right)}{f\left(\overrightarrow{\theta}\right) f\left(\text{data} \middle| \overrightarrow{\theta}\right)} \times \frac{q\left(\overrightarrow{\theta} \middle| \overrightarrow{\theta}^*, \text{data}\right)}{q\left(\overrightarrow{\theta}^* \middle| \overrightarrow{\theta}, \text{data}\right)}$$
     - 2.1 If greater than a random draw from a standard uniform distribution, set $\overrightarrow{\theta} = \overrightarrow{\theta}^*$
     - 2.2 Otherwise, retain $\overrightarrow{\theta} = \overrightarrow{\theta}$
  3. Optionally store $\overrightarrow{\theta}$ after some warmup period
- Gibbs sampler
  - Can be seen as a special case of M-H
  - Update one parameter at a time; cycle through parameters
  - Specify $q\left(\theta_i^* \middle| \overrightarrow{\theta}_{-i}, \text{data}\right)$ as full-conditional PDF of *i*th parameter, given all other parameters $\overrightarrow{\theta}_{-i}$
  - Critical ratio always equals 1 so always accept proposals

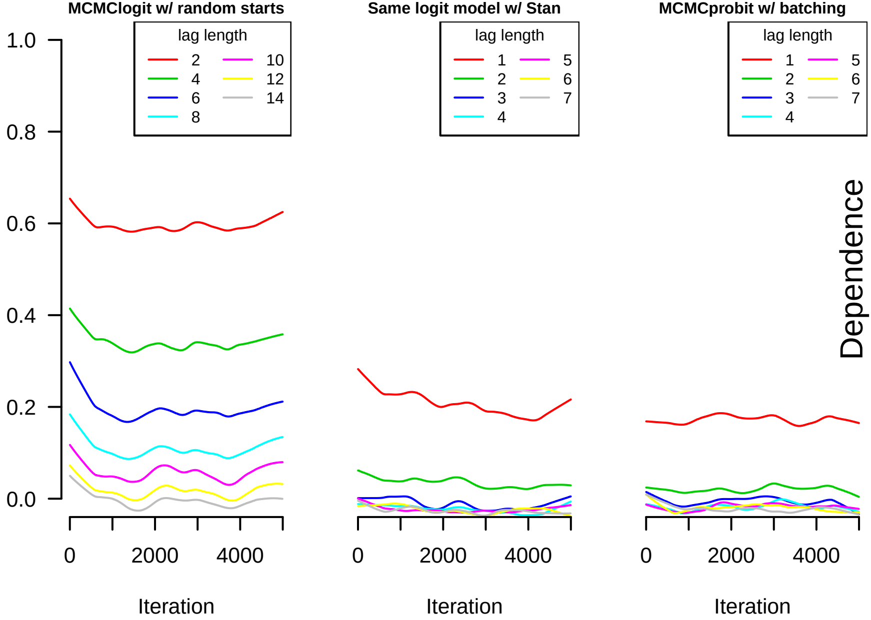
Weaknesses of Metropolis-Hastings and Gibbs

- For M-H, a lot hinges on choice of $q\left(\overrightarrow{\theta}^*\middle|\overrightarrow{\theta},\text{data}\right)$
  - For “easy” jumping distributions, sampler randomly walks
    - Difficult to get to stationary distribution
    - Difficult to get into and out of tails of distribution
  - Tradeoff between acceptance probability and long jumps
    - High acceptance prob. $\Longrightarrow$ short jumps & high dependence
    - Long jumps $\Longrightarrow$ low acceptance prob. & high dependence
- For Gibbs, the problems are a bit different
  - $q\left(\theta_i^*\middle|\overrightarrow{\theta}_{-i},\text{data}\right)$ can be hard to derive analytically
  - Conditional variance may be much smaller than marginal variance, which implies $\theta_i^* \approx \theta_i$
  - Thus, consecutive draws can have high dependence

Hamiltonian Monte Carlo (HMC) and Stan

- Very clever symmetric $q\left(\overrightarrow{\theta}^{\,*} \middle| \overrightarrow{\theta}, \text{data}\right) = q\left(\overrightarrow{\theta} \middle| \overrightarrow{\theta}^{\,*}, \text{data}\right)$
    - Based on metaphor of Hamiltonian dynamics
    - Solves Ordinary Differential Equations for $\overrightarrow{\theta}^{\,*}$
    - Allows long jumps with high acceptance probability
- Weaknesses of Hamiltonian Monte Carlo
    - Also need to compute $\nabla \overrightarrow{\theta}$ (which is hard or slow)
    - Need to tune $q\left(\overrightarrow{\theta}^{\,*} \middle| \overrightarrow{\theta}, \text{data}\right)$ during warmup period
- Strengths of Stan, which is a variant of HMC
    - Computes $\nabla \overrightarrow{\theta}$ via automatic differentiation
    - Self-tuning, although you can also do it manually

Posterior Dependence in Low Birthweight Example



Principles Needed to Use Stan Effectively

- In HMC, $\frac{q\left(\vec{\theta}\middle|\vec{\theta}^*,\text{data}\right)}{q\left(\vec{\theta}^*\middle|\vec{\theta},\text{data}\right)} = 1$ so do not worry about that
- User needs to express in the Stan language $\ln\left(f\left(\vec{\theta}\right)\times f\left(\text{data}|\ \vec{\theta}\right)\right) = \ln\left(f\left(\vec{\theta}\right)\right) + \ln\left(f\left(\text{data}|\ \vec{\theta}\right)\right)$
- Can ignore terms that do not depend on $\vec{\theta}$
- You need to declare the support of $\vec{\theta}$
- HMC is vulnerable to varying parameter scales (some really big, others really small)
  - Stan mitigates this somewhat with tuning
  - User can mitigate it a lot with rescaling
- Stan is vulnerable to non-constant parameter dependence
  - Try to respecify your model in an equivalent way that reduces or regularizes the parameter dependence

Steps of Stan

1. You write the model in (text) .stan file w/ R-like syntax
2. The parser, stanc, does two things
   - checks that your model is valid
   - writes a conceptually equivalent C++ source file
3. C++ compiler creates a binary file from the C++ source
4. You execute the binary from R (or Python / command-line)
5. You analyze the resulting samples from the posterior

Types

- Primitive scalar types: `real` and `int`
- (column) `vector[K]` of K reals w/ 4 constrained subtypes
  - `simplex[K]` (non-negative and sums to 1)
  - `unit_vector[K]` (sum of squares equals 1)
  - `ordered[K]` (each element is greater than the previous)
  - `positive_ordered[K]` (and all are positive)
- `row_vector[K]` of reals
- `matrix[N,K]` of reals w/ 3 constrained subtypes
  - `cov_matrix[K]` (covariance matrix, or its inverse)
  - `cholesky_factor_cov[K,K]` (Cholesky factor thereof)
  - `corr_matrix[K]` (correlation matrix)
  - `cholesky_factor_corr[K]` (Cholesky factor thereof)
- `real`, `int`, `vector` (plain), `row_vector`, and `matrix` (plain) can have lower and / or upper bounds inside <>
- Can have homogenous arrays of any of the above, e.g. `row_vector<lower=0,upper=1>[K] p[N];`

The `data` Block of a .stan File

- Contains everything passed from R to Stan
- Can be modeled data (**y**), covariates (**X**), constants (K)
- Basically, everything posterior distribution conditions on
- Can have comments in R style (#) or C++ style (// or /* */)

```
data {
  int<lower=1> K;    # number of covariates
  int<lower=1> N;    # number of observations
  matrix[N,K] X;     # predictor matrix
  real y[N];         # outcome variable

  // stuff for informative priors in regression
  vector[K] beta_0;
  cov_matrix[K] A_0;
  real<lower=0> v_0; /* or could be int */
  real<lower=0> s2_0;
}
```

Optional `transformed data` Block of a .stan File

- Is executed only once before the sampling iterations
- Can be used to calculate needed functions of data
- Not so necessary if calling Stan from R
- I often use it to check that data was passed correctly
- Need to declare objects before they are assigned (<-)

```
transformed data {
  cov_matrix[K] XtXpA_0;
  XtXpA_0 <- crossprod(X) + A_0;
  print("K =", K);
  print("N =", N);
}
```

The `parameters` Block of a .stan File

- Declare everything whose posterior distribution is sought
- Cannot declare any integer parameters currently, only real
- Must specify the support of the parameters
- Stan is really sampling from unbounded parameter space
  - Behind-the-scenes transformations to yield parameters
  - For example, `exp()` of an unbounded yields a positive
  - Jacobians of these transformations handled automatically

```
parameters {
  vector[K] beta;        # unrestricted
  real<lower=0> sigma2; # restricted to be >= 0
  /* legal to have lower and / or upper bounds
  depend on the values of previously-declared
  parameters */
}
```

The optional `transformed parameters` Block

- Similar in structure to the `transformed data` block
- But is executed every iteration (and leapfrog step)
- Used to calculate deterministic functions of parameters
- Need to declare objects before they are assigned
- Such objects can then be used in the `model` block
- Constraints are validated and samples are stored

```
transformed parameters {
  real<lower=0> sigma;
  sigma <- sqrt(sigma2);
}
```

The `model` Block of a .stan File

- Can declare more objects and then assign them
- Constraints are not validated and samples not stored
- Used to add log-priors and log-likelihood w/ $\sim$ statements
- Can also manually increment the log-posterior

```
model {
  y ~ normal(X * beta, sigma); # log-likelihood
  beta ~ multi_normal_prec(beta_0, XtXpA_0);
  sigma2 ~ inv_gamma(v_0, s2_0);
}
```

The `generated quantities` Block of a .stan File

- Only evaluated for non-thinned post-warmup iterations
- Can declare more objects and then assign them
- Constraints are not validated but samples are stored
- Cannot reference anything in the `model` block
- Primarily used for
  - Interesting functions of posterior that don't go into likelihood
  - Posterior predictive distributions

```
generated quantities {
  vector[N] y_tilde;
  for (i in 1:N) {
    y_tilde[i] <- normal_rng(row(X,i) * beta,
                             sigma);
  }
}
```

Examples

- In RStan, any example can be executed with

```
posterior <- stan_demo() # choose example
```

- Also, can browse repo at https://github.com/stan-dev/stan/tree/develop/src/models for same set of examples. Be sure to download
  - foo.stan file
  - foo.data.R file (invoke `pystan.misc.read_rdump("/path/to/foo.data.R")` to use with PyStan)
- Many examples in Stan manual at http://mc-stan.org
- Examples today are available from https://github.com/stan-dev/rstan/blob/develop/StanNYCMeetup.zip