How to Use (R)Stan to Estimate Models in External R Packages

Ben Goodrich of Columbia University (benjamin.goodrich@columbia.edu)

July 6, 2017

Obligatory Disclosure

- Ben is an employee of Columbia University, which has received several research grants to develop Stan
- Ben is also a manager of GG Statistics LLC, which provides support, consulting, etc. for businesses using Stan
- According to Columbia University policy, any such employee who has any equity stake in, a title (such as officer or director) with, or is expected to earn at least $\$5,000.00$ per year from a private company is required to disclose these facts in presentations

Four Key Points

1. Numerical differentiation is worse than automatic differentiation (AD)
2. Optimization is worse than Markov Chain Monte Carlo (MCMC) sampling
3. Gibbs and especially Metropolis-Hastings sampling are worse than No U-Turn Sampling (NUTS) for differentiable posterior distributions
4. It is easy to build an R package that utilizes Stan's implementation of NUTS

What Is Stan?

- Includes a new computer language for expressing statistical models
- Includes a translator of high-level Stan syntax to somewhat low-level C++
- Includes a matrix and scalar math library that supports autodifferentiation
- Includes new (and old) gradient-based algorithms for statistical inference
- Includes interfaces from R and other high-level software
- Includes (not Stan specific) post-estimation R functions
- Includes a large community of users and many developers

Who Is Using (R)Stan?

- Stan is being used in academia, business, and government by people who want to estimate good models
- 37 R packages reverse {depends, imports, linking to, suggests} rstan
- Notable download rates
  - rstan: downloads 10K/month
  - rstanarm: downloads 2954/month
  - brms: downloads 2294/month
  - prophet: downloads 2323/month
- Stan is used for fitting climate models, clinical drug trials, genomics and cancer biology, population dynamics, psycholinguistics, social networks, finance and econometrics, professional sports, publishing, recommender systems, educational testing, and many more.

Numerical Differentiation Is Worse than AD

- Numerical differentiation of a function f is essentially, for small h,

$$\frac{\partial}{\partial \theta_k} f(\boldsymbol{\theta}) \approx \frac{f(\boldsymbol{\theta}) - f(\boldsymbol{\theta} + \boldsymbol{h} \times \mathbf{e}_{\boldsymbol{k}})}{h}$$

- Suffers from cancellation *by construction*, so it may not accurately approximate $\frac{\partial}{\partial \theta_k} f(\boldsymbol{\theta})$ and you cannot tell whether it is accurate from the output
- Automatic Differentiation does not suffer from cancellation and is about as numerically accurate as f is
- AD depends on each function knowing its partial derivatives and instructing the C++ compiler to evaluate the chain rule automatically
- The Stan Math Library is arguably the best AD implementation for statistical inference problems

Use Stan to Minimize $(\mathbf{x} - \mathbf{a})^\top (\mathbf{x} - \mathbf{a})$ in C++

```cpp
// [[Rcpp::depends(BH)]]
// [[Rcpp::depends(RcppEigen)]]
// [[Rcpp::depends(StanHeaders)]]
#include <Rcpp.h>
#include <RcppEigen.h>
#include <stan/math.hpp>  // pulls in everything; could be more specific with included headers

// [[Rcpp::export]]
double f(Eigen::VectorXd x, Eigen::VectorXd a) {  // objective function in doubles
  return stan::math::dot_self( (x - a).eval() );  // dot_self() is a dot product with the same vector
}
stan::math::var f(Eigen::Matrix<stan::math::var, Eigen::Dynamic, 1> x, Eigen::VectorXd a) {
  return stan::math::dot_self( (x - stan::math::to_var(a)).eval() );  // same but with vars
}
// [[Rcpp::export]]
std::vector<double> g(Eigen::VectorXd x, Eigen::VectorXd a) {  // gradient by AD using Stan
  auto x_var = stan::math::to_var(x); std::vector<stan::math::var> theta; std::vector<double> grad;
  for (int k = 0; k < x.rows(); k++) theta.push_back(x_var.coeff(k));
  stan::math::var lp = f(x_var, a); lp.grad(theta, grad); return grad;
}
```

```r
optim(rnorm(3), fn = f, gr = g, a = c(1, 2, 3), method = "BFGS")$par  # Rcpp exported f and g
```

```
## [1] 1 2 3
```

Same Thing Using the Stan Language

```
data {
  int<lower=0> K; // equals 3 in this example
  vector[K] a;
}
parameters { vector[K] x; }
model { target += -dot_self(x - a); } // note scaling by -1
```

```
library(rstan)  # a compiled object called sm has been created by knitr
optimizing(sm, data = list(K = 3, a = c(1, 2, 3)), as_vector = TRUE)$par
```

```
## Initial log joint probability = -18.7217
## Optimization terminated normally:
##   Convergence detected: gradient norm is below tolerance
```

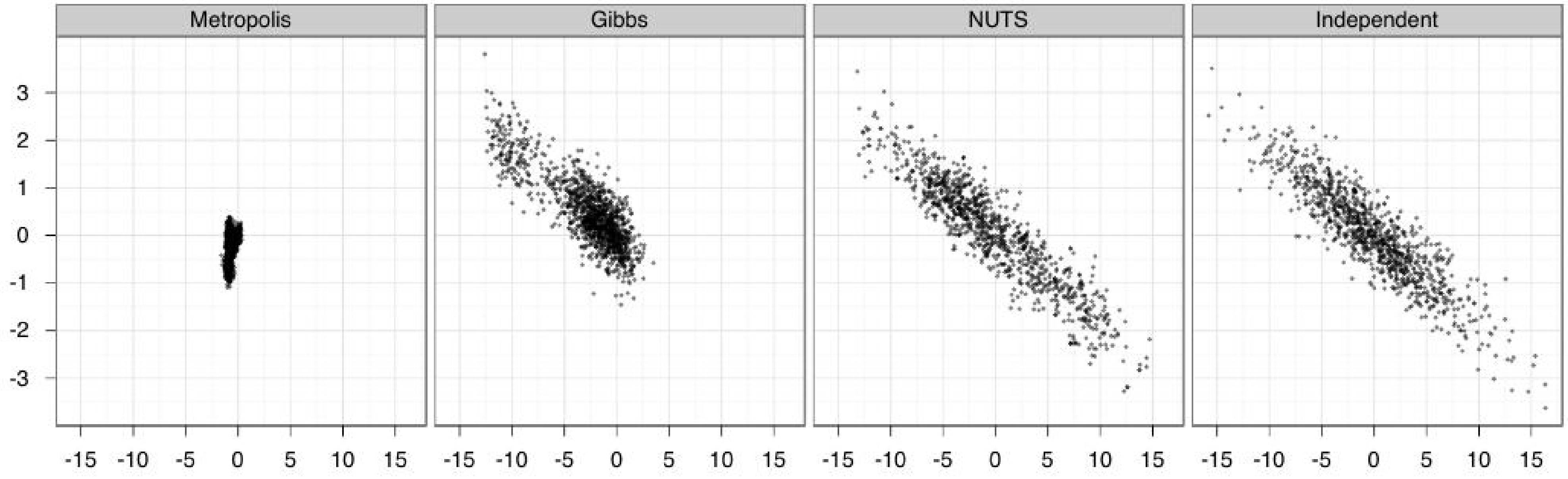
```
## x[1] x[2] x[3]
##    1    2    3
```

But Do Not Do This in Your Own R Packages

- You generally (the **brms** package is the exception) do not want to make a Stan-based R package that mimics the **rstan** user experience:
  1. Requires users to have a C++ toolchain (RTools on Windows, Xcode on Mac)
  2. Requires users to wait 30+ seconds to compile a Stan program at runtime
  3. Can't do 5 second examples or 2 minutes of unit tests in that case
- Stata 15 allows you to put the `bayes` keyword in front of almost any core estimation command to obtain draws from its posterior using a combination of Metropolis and Gibbs sampling
- We want the same thing for R, except correct

Example of Drawing from a Multivariate Normal

- $\mathbf{y} \sim \mathcal{N}_{250}\left(\mathbf{0}, \mathbf{\Sigma}\right)$ where $\mathbf{\Sigma}$ is ill-conditioned but focus on just 2 dimensions
- Do 1 million draws w/ Random Walk Metropolis & Gibbs, thinning by 1000
- Do 1000 draws with the NUTS algorithm in Stan and 1000 independent draws



Comparison of MCMC Samplers

Wrappers for Stan Programs

- Can use Stan for estimation without personally knowing the Stan language
- The **rstanarm** R package provides interface to a handful of pre-compiled Stan programs whose syntax mirrors that of popular model-fitting functions in R:
    - `lm`, `aov`, and `glm`
    - `MASS::polr` and `MASS::glm.nb`
    - `betareg::betareg`
    - `lme4::lmer`, `lme4::glmer`, and `lme4::nlmer`
    - `gamm4::gamm4`
    - `survival::clogit`
- Other packages that provide pre-compiled Stan models include: **beanz**, **bmlm**, **breathteststan**, **dfpk**, **eggCounts**, **gastempt**, **idem**, **MADPop**, **survHE**, **treatSens**, and **walker**

Chocolate Cake Example

```r
library(rstanarm); cake <- lme4::cake; cake$angle10 <- cake$angle / 10
post <- stan_gamm4(angle10 ~ recipe + s(temp, k = 5), # parsed by mgcv
                   data = cake, family = Gamma(link = "log"), random = ~(1 | replicate/recipe), # parsed by lme4
                   seed = 123, init_r = 0.5, adapt_delta = 0.995); plot_nonlinear(post)
```

Regression Example with Student t Errors

```
data {
  int<lower=0> N; int<lower=0> K; matrix[N,K] X; vector[N] y;
  real<lower=0> shape; real<lower=0> rate; // hyperparameters
}
parameters {
  real alpha;             // intercept
  vector[K] beta;         // coefficients
  real<lower=0> s;        // scale
  real<lower=0> d_raw;    // primitive for degrees of freedom
}
transformed parameters {
  real d = d_raw / rate; // degrees of freedom
}
model {
  target += student_t_lpdf(y | d, alpha + X * beta, s); // likelihood
  target += gamma_lpdf(d_raw | shape, 1); // implies d ~ gamma(shape, rate)
}
```

Creating an R Package that Uses (R)Stan

```r
rstantools::rstan_package_skeleton("treg", force = TRUE, stan_files = "treg.stan")
dir("treg", recursive = TRUE)
```

```
##  [1] "cleanup"
##  [2] "cleanup.win"
##  [3] "DESCRIPTION"
##  [4] "exec/treg.stan"
##  [5] "inst/chunks/common_functions.stan"
##  [6] "inst/chunks/license.stan"
##  [7] "man/treg-package.Rd"
##  [8] "NAMESPACE"
##  [9] "R/stanmodels.R"
## [10] "R/treg-internal.R"
## [11] "R/zzz.R"
## [12] "Read-and-delete-me"
## [13] "src/Makevars"
## [14] "src/Makevars.win"
## [15] "tools/make_cpp.R"
```

Writing the R Wrapper

- Put something like this into treg/R/treg.R

```r
treg <- function(formula, data, shape = 2, rate = 0.1, ...) { # choose good defaults
  mf <- lm(formula, data, method = "model.frame")
  X <- model.matrix(formula, data = mf)
  if (colnames(X)[1] == "(Intercept)") X <- X[,-1, drop = FALSE]
  y <- model.response(mf, type = "numeric")
  standata <- list(N = NROW(X), K = NCOL(X), X = X, y = y, shape = shape, rate = rate)
  post <- rstan::sampling(stanmodels$treg, data = standata, ...)
  out <- list(stanfit = post) # further process the output presumably
  class(out) <- "treg"
  return(out)
}
```

- Read (and delete) the Read-and-delete-me file for more details
- Then call `devtools::install(args = "--preclean")` to install it
- See https://cran.r-project.org/package=rstantools/developer-guidelines.html

Where to Start / Get Help?

- RStanArm vignettes: https://cran.r-project.org/package=rstanarm
- RStanTools vignette: https://cran.r-project.org/package=rstantools/developer-guidelines.html
- User manual: http://mc-stan.org/documentation/
- Examples of Stan programs: https://github.com/stan-dev/example-models
- Mailing list: http://discourse.mc-stan.org
- Stan conference: http://mc-stan.org/events/stancon2018/